# Graph Transformers


Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
*ojlee@catholic.ac.kr*

네트워크 과학 연구실 NETWORK SCIENCE LAB

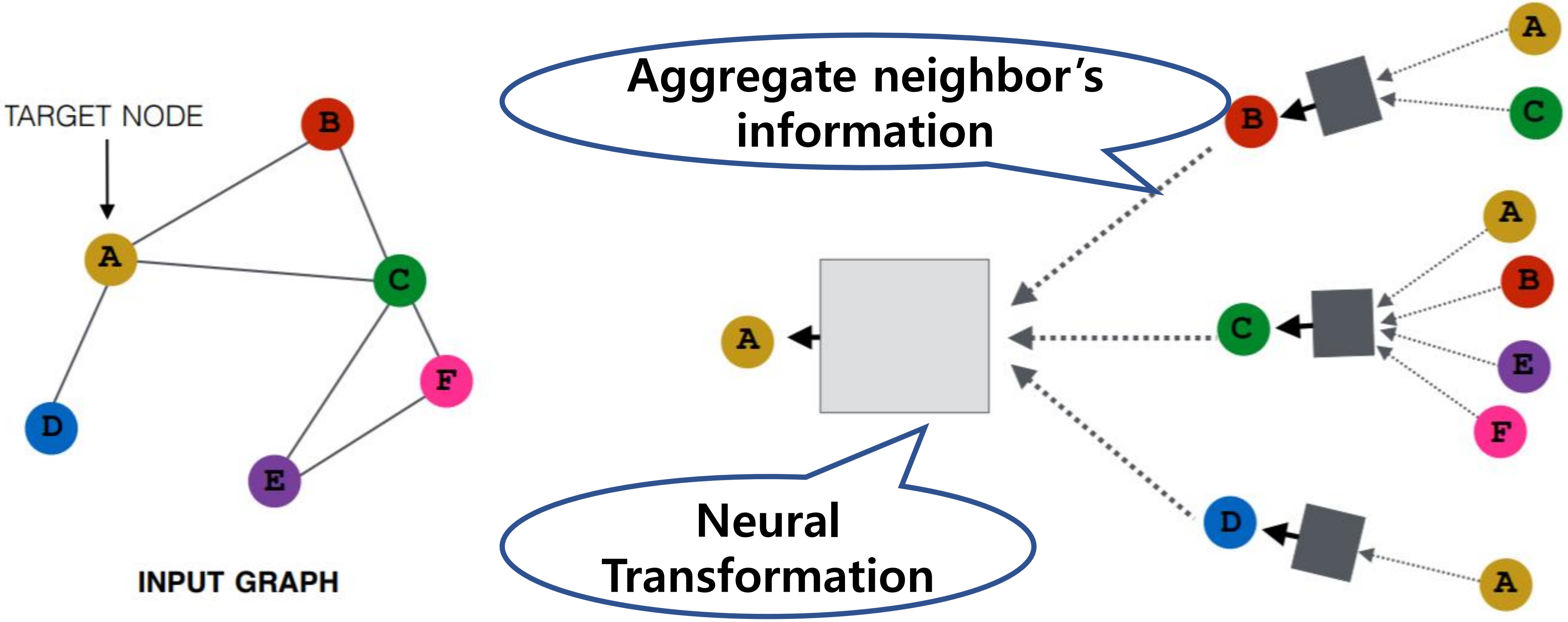
가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA


# Contents

- From MPNNs to Self-attention
- Positional Encoding in Graphs
- Transformers are Graph Neural Networks
- Representative Transformer models

# Graph Neural Networks (GNNs)

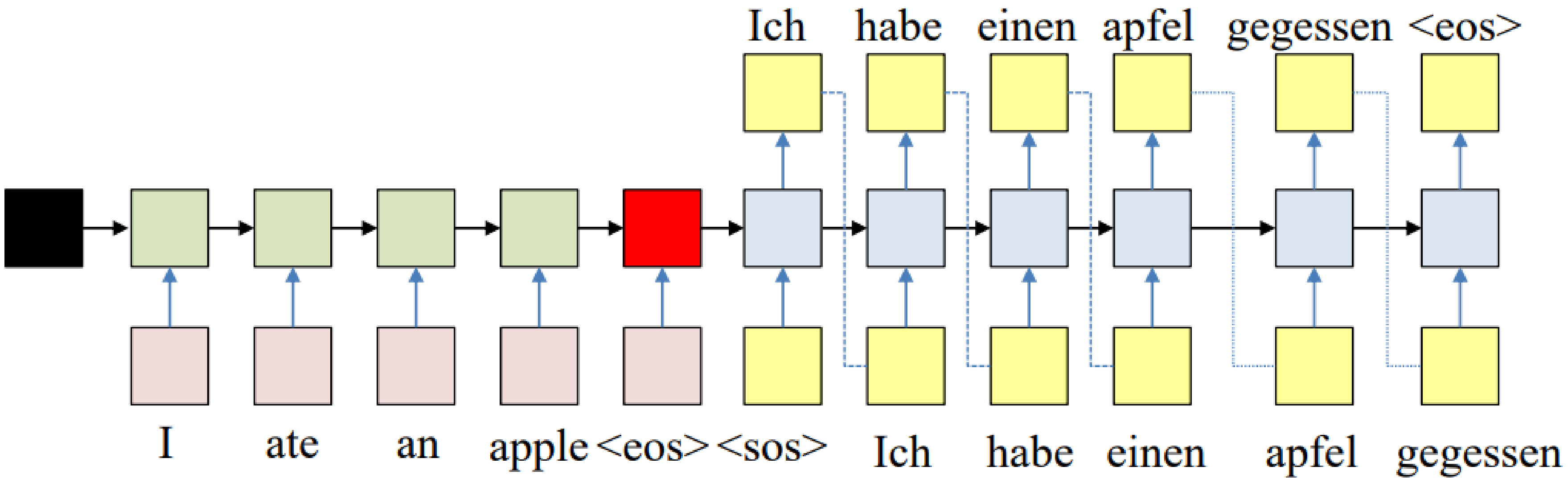
- **Key Idea:** Each node aggregates messages from its neighborhood to get contextualized node embedding.
- **Limitation:** Most GNNs focus on homogeneous graph.



# Recap: Seq2Seq models

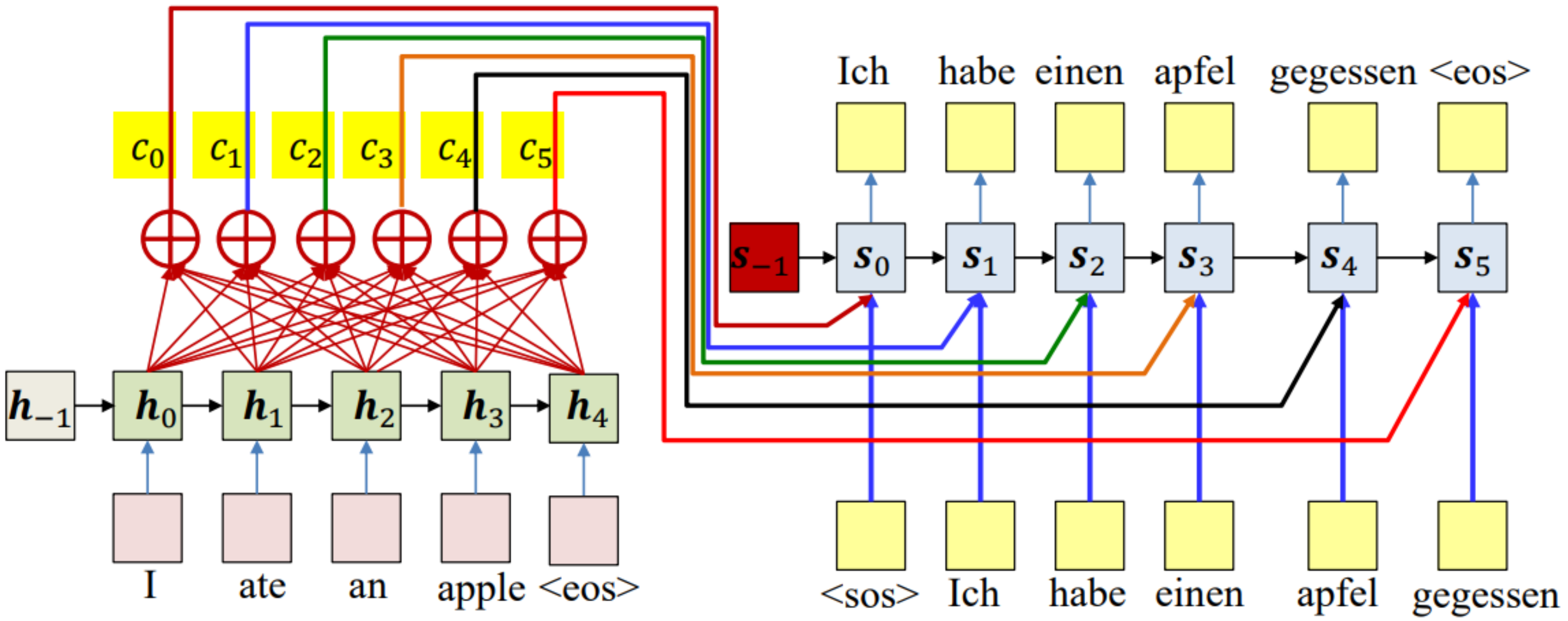


- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a second RNN uses the hidden activation as initial state to produce a sequence of outputs



# Recap: Attention Models

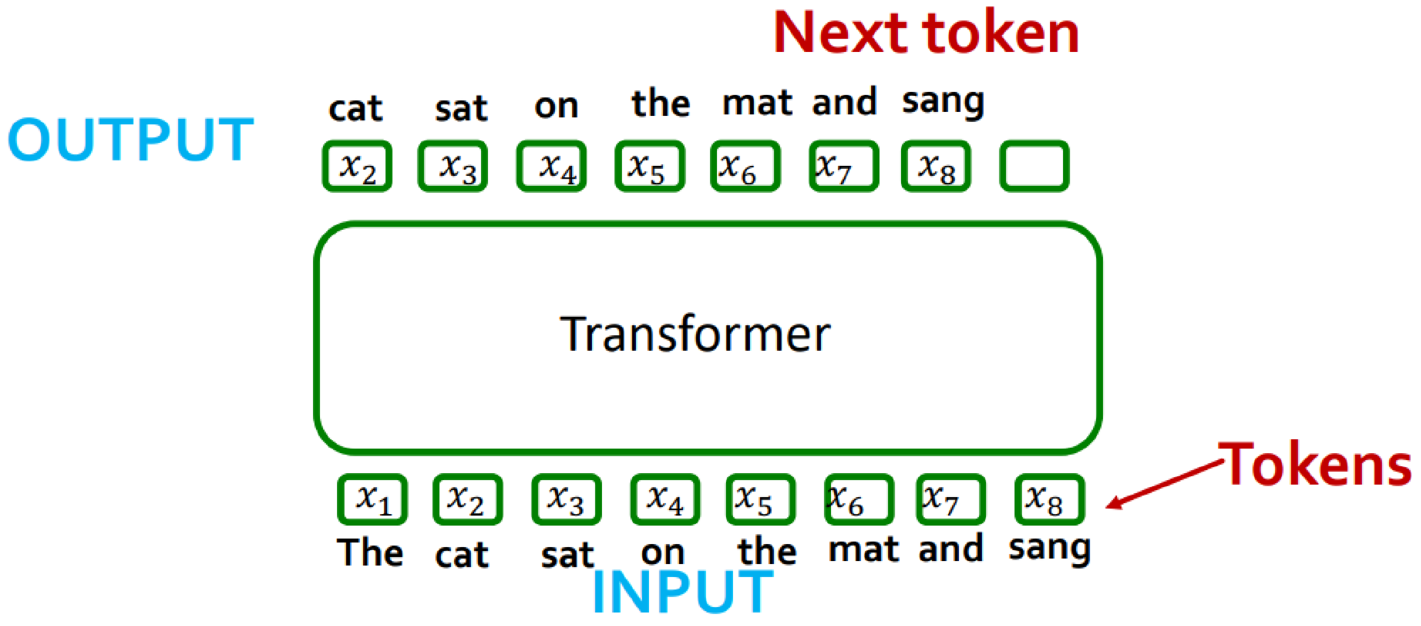
- Encoder recurrently produces hidden representations of input word sequence
- Decoder recurrently generates output word sequence
  - For each output word the decoder uses a weighted average of the hidden input representations as input “context”, along with the recurrent hidden state and the previous output word



# Recap: Transformers



- Transformer ingest **TOKENS**
- Transformers map 1D sequences of vectors to 1D sequences of vectors known as tokens
  - Tokens describe a ”piece” of data – e.g., a word
- What output sequence?
  - Option 1: next token => GPT



# Recap: Self attention

- First, for every word in the input sequence we compute an initial representation
  - E.g. using a single MLP layer
- Then, from each of the hidden representations, we compute a query, a key, and a value.
  - Using separate linear transforms
  - The weight matrices $Wq$, $Wk$ and $Wv$ are learnable parameters

- The updated representation for the word is the attention-weighted sum of the values for all words (Including itself)

$$q_i = \boldsymbol{W}_q h_i$$

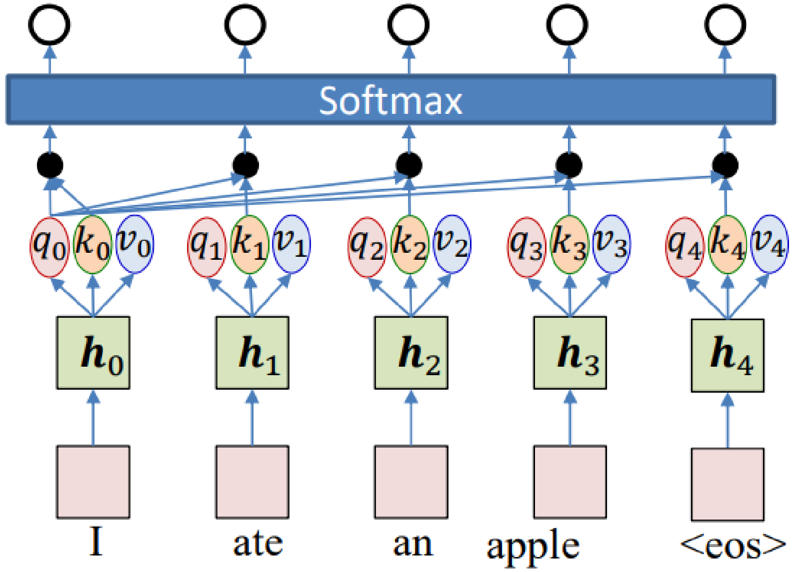
$$k_i = \boldsymbol{W}_k h_i$$

$$v_i = \boldsymbol{W}_v h_i$$

$$w_{ij} = attn(q_i, k_{0:N})$$

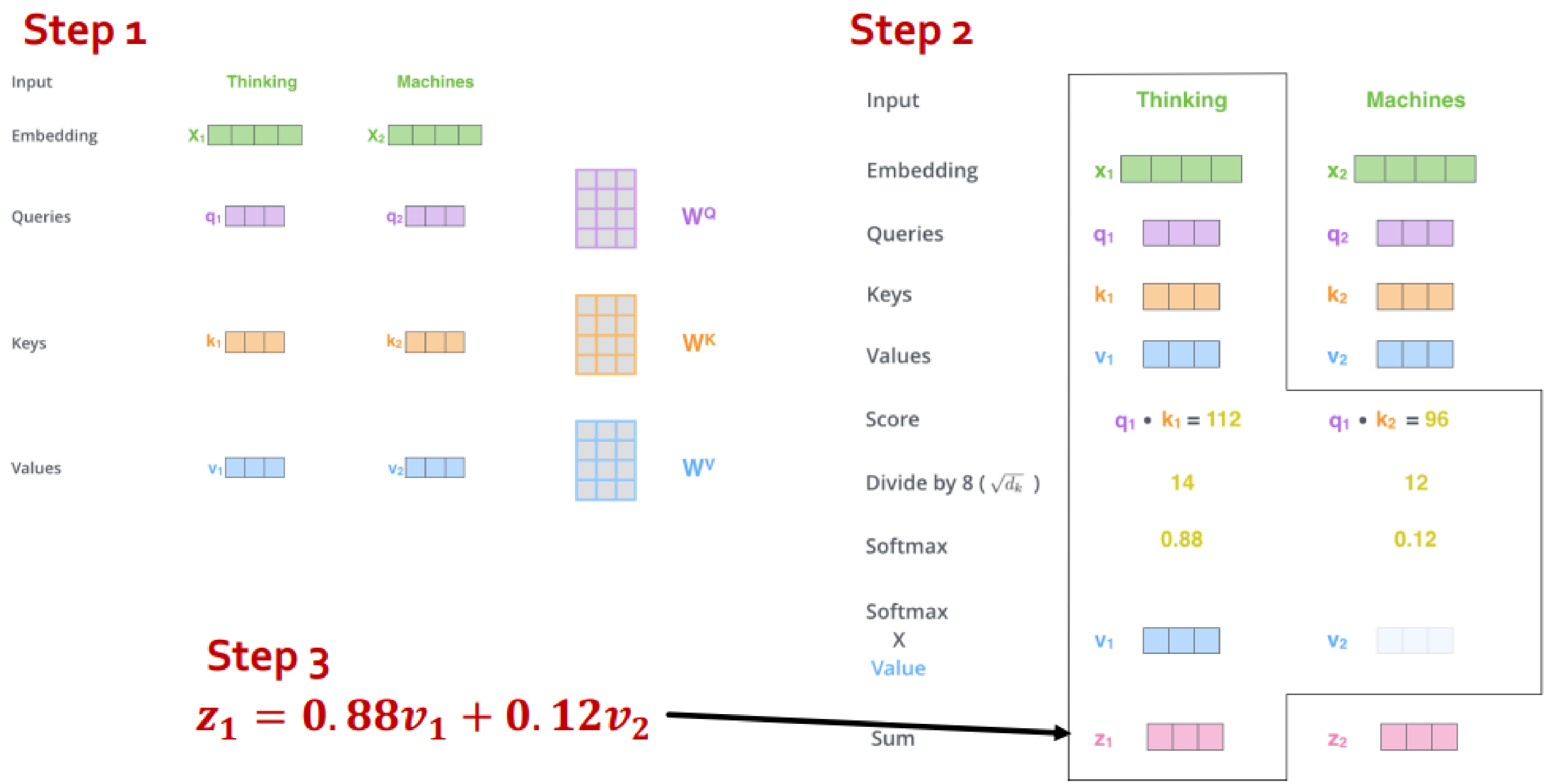
$$e_{ij} = q_i^T k_j$$

$$w_{i0}, \dots, w_{iN} = softmax(e_{i0}, \dots, e_{iN})$$



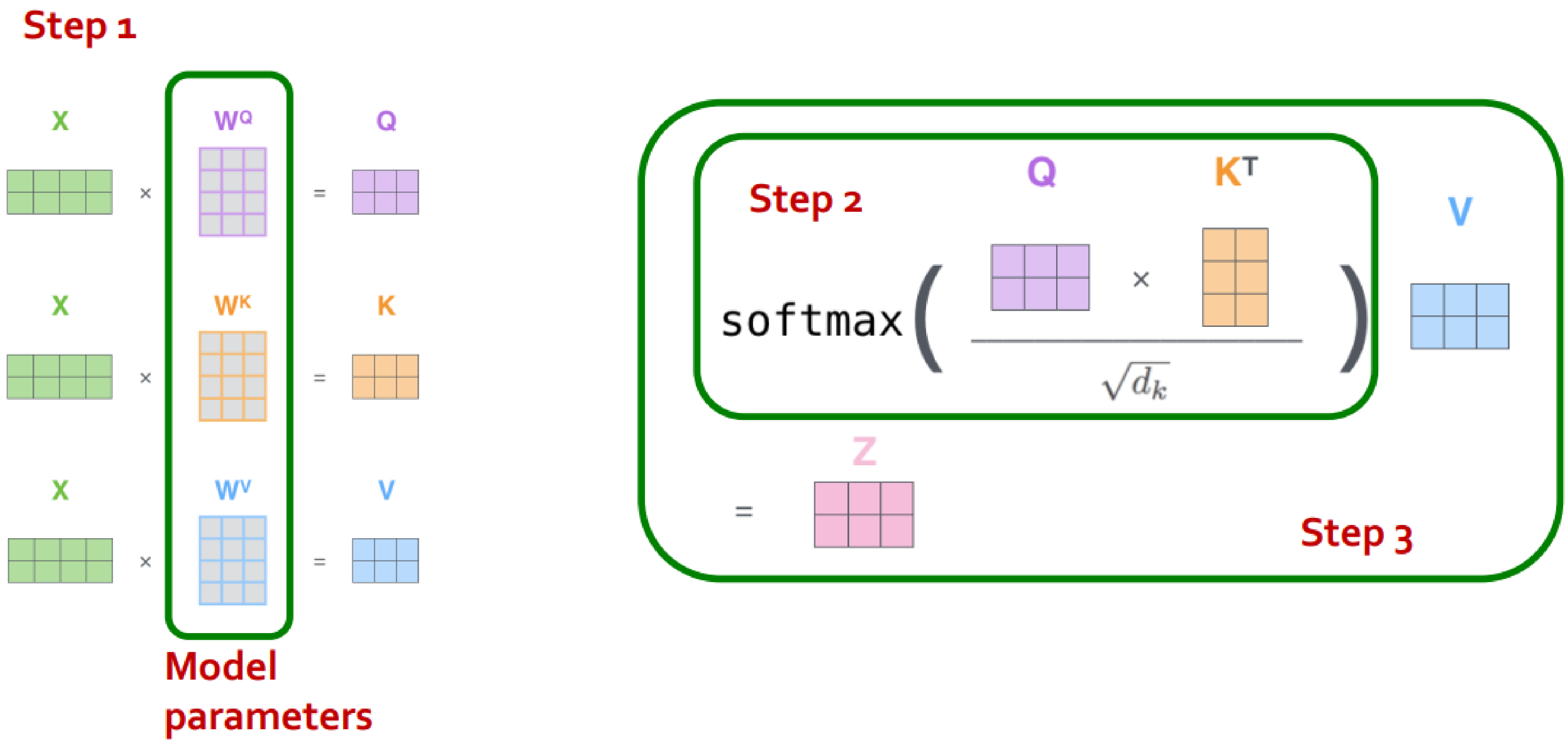
# Recap: Self attention

- Step 1: compute "key, value, query" for each input
- Step 2 (just for $x_1$): compute scores between pairs, turn into probabilities (same for $x_2$)
- Step 3: get new embedding $z_1$ by weighted sum of $v_1$, $v_2$



# Recap: Self attention

- Same calculation in matrix form




https://jalammar.github.io/illustrated-transformer/

# Recap: Multi-head Self attention



- We can have multiple such attention “heads”
  - Each will have an independent set of queries, keys and values
  - Each will obtain an independent set of attention weights
    - Potentially focusing on a different aspect of the input than other heads
  - Each computes an independent output
- The final output is the concatenation of the outputs of these attention heads
- “MULTI-HEAD ATTENTION”

(actually Multi-head self attention)

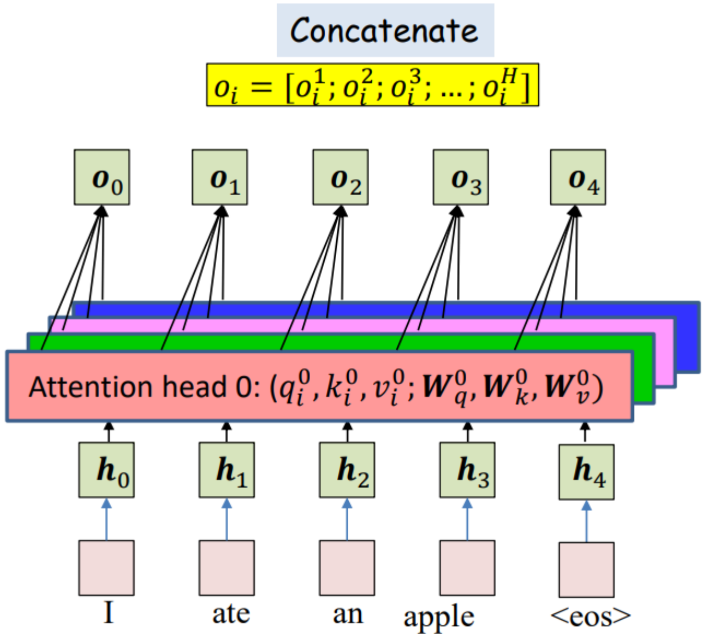
$$q_i^a = \boldsymbol{W}_q^a h_i$$

$$k_i^a = \boldsymbol{W}_k^a h_i$$

$$v_i^a = \boldsymbol{W}_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$



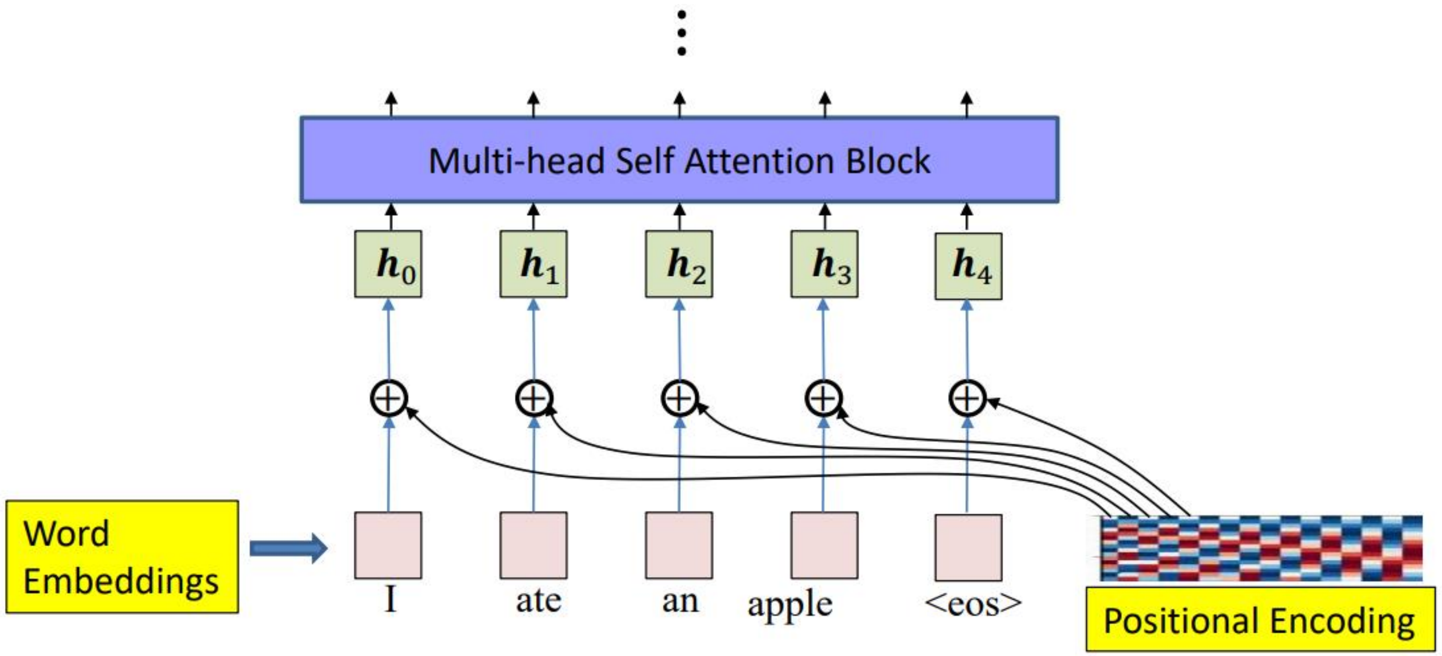
# Recap: Positional Encoding



- Positional Encoding: A sequence of vectors $P_0, P_1, \cdots, P_N$ to encode position
  - Every vector is unique (and uniquely represents time)
  - Relationship between $P_t$ and $P_{t+k}$ only depends on the distance between them:

$$P_{t+k} = M_k P_t$$

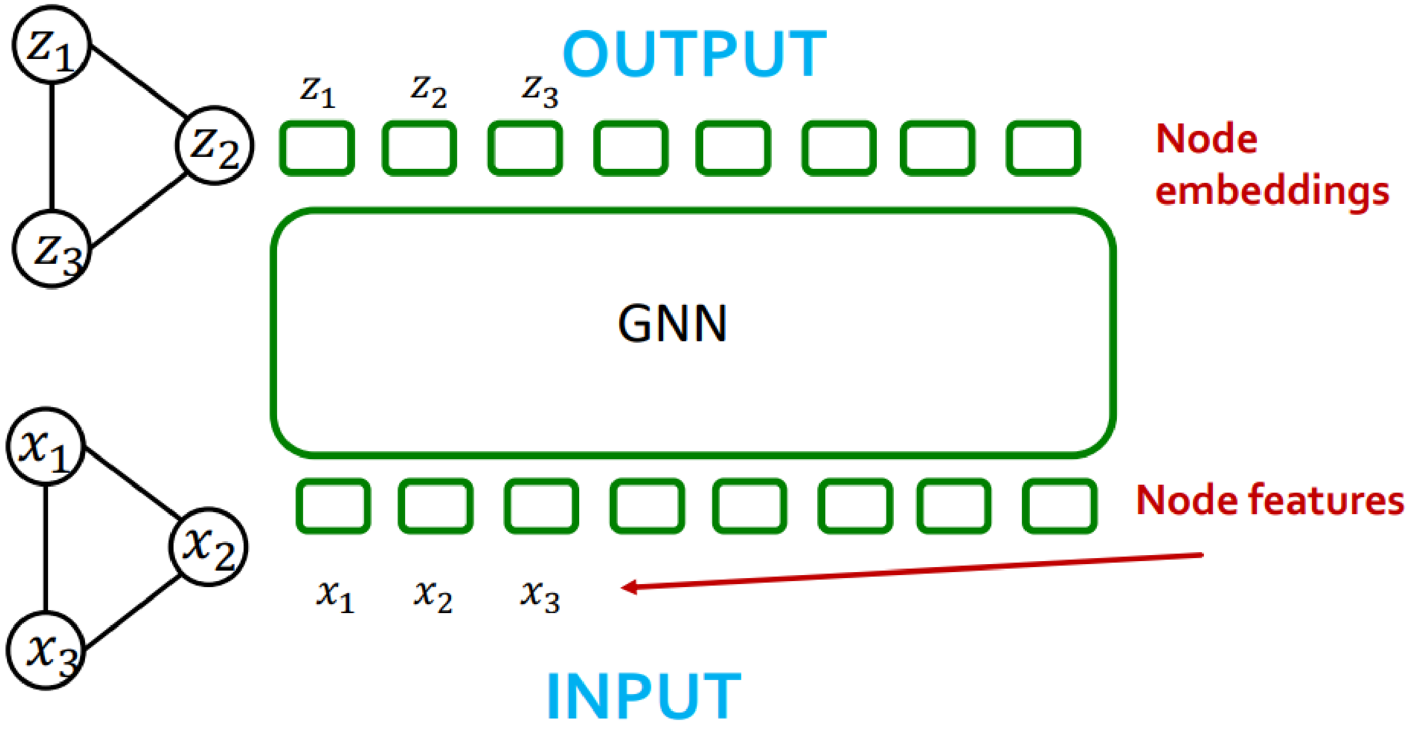
- The linear relationship between $P_t$ and $P_{t+k}$ enables the net to learn shiftinvariant "gap" dependent relationships



# Comparing Transformer and GNNs



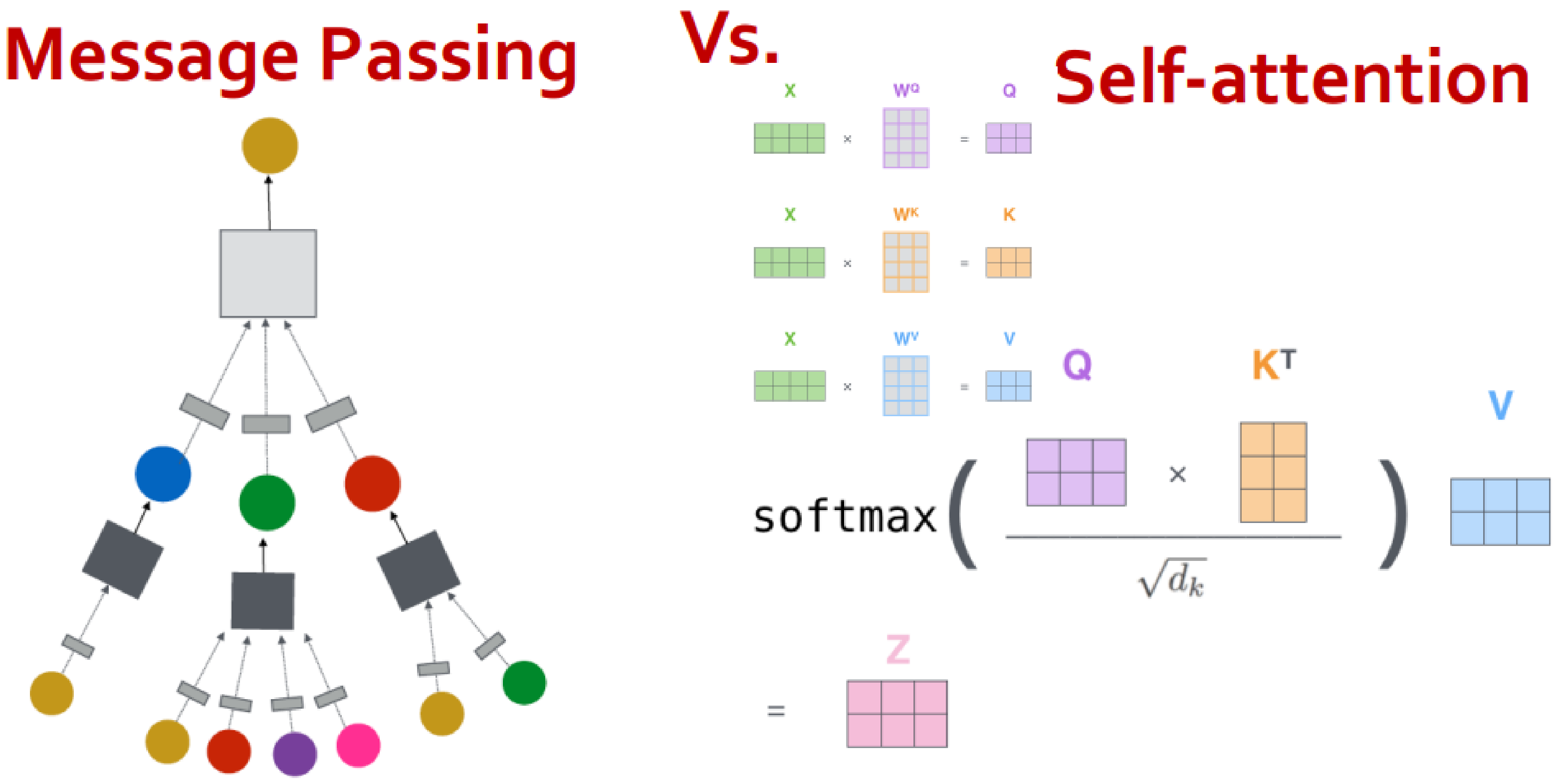
- **Similarity**: GNNs also take in a sequence of vectors (in no particular order) and output a sequence of embeddings
- **Difference**: GNNs use message passing, Transformer uses self-attention



# Comparing Transformer and GNNs

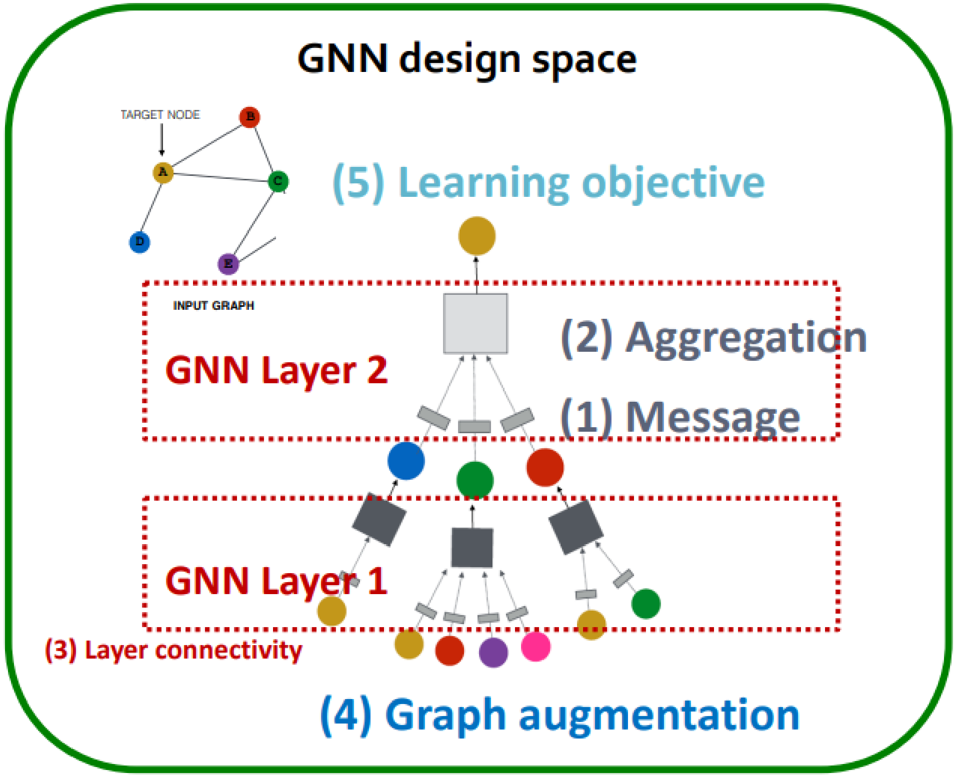


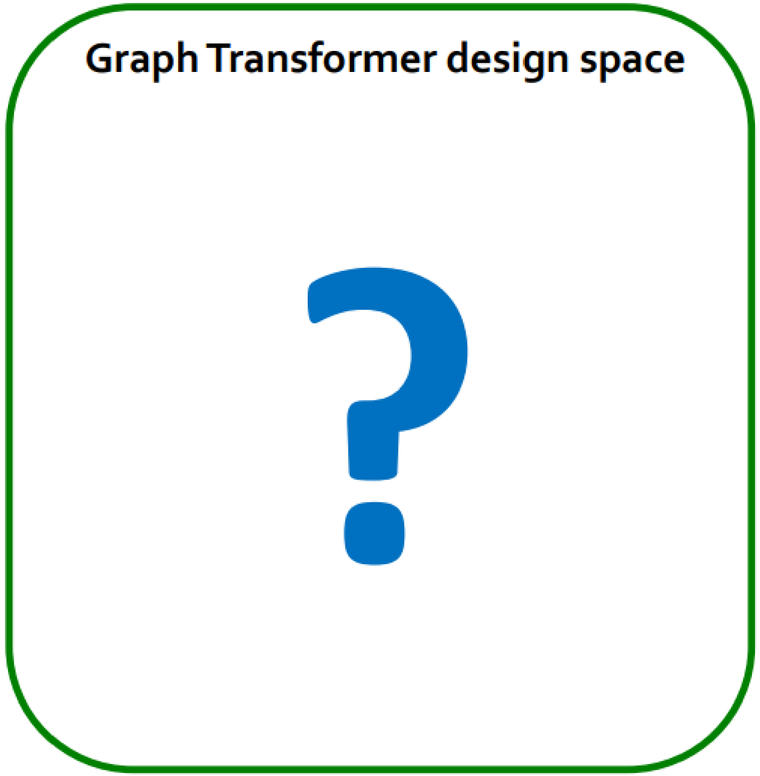
- **Difference**: GNNs use message passing, Transformer uses self-attention
- Are self-attention and message passing really different?



# Comparing Transformer and GNNs

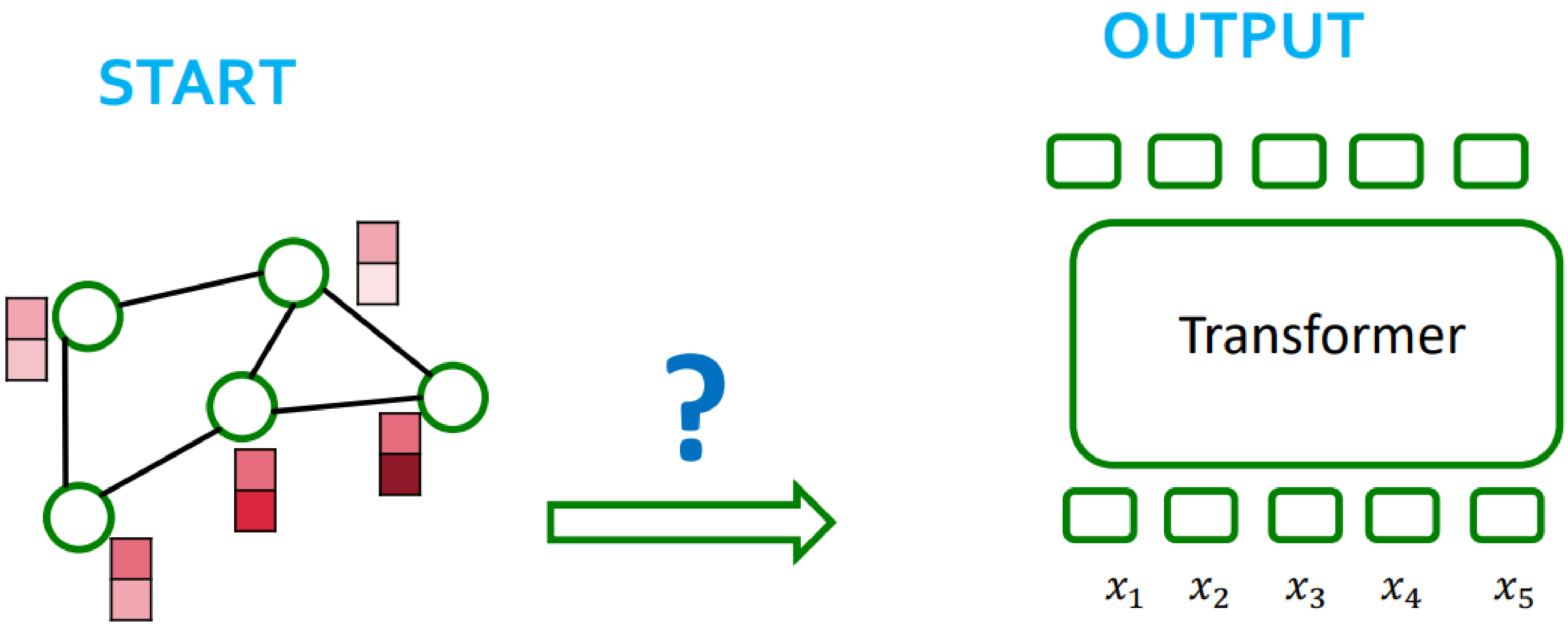
- We know a lot about the design space of GNNs
- What does the corresponding design space for Graph Transformers look like?



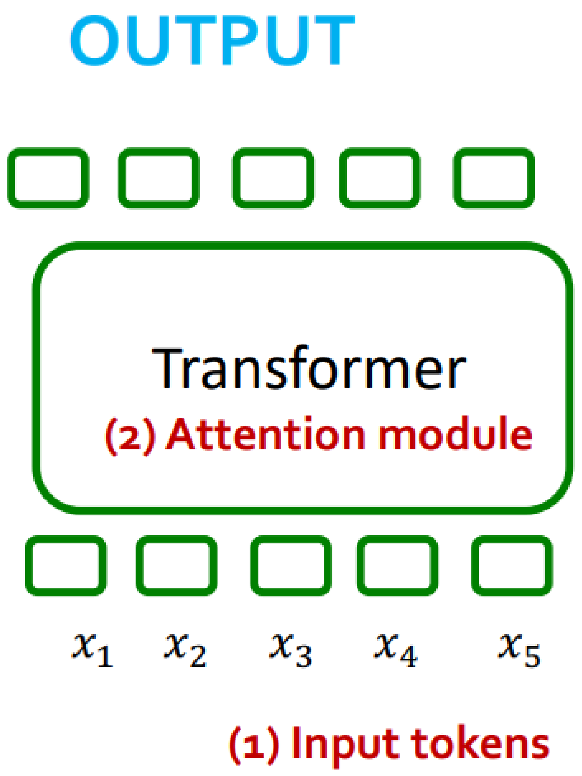
# Processing Graphs with Transformers

- We start with graph(s)
- How to input a graph into a Transformer?



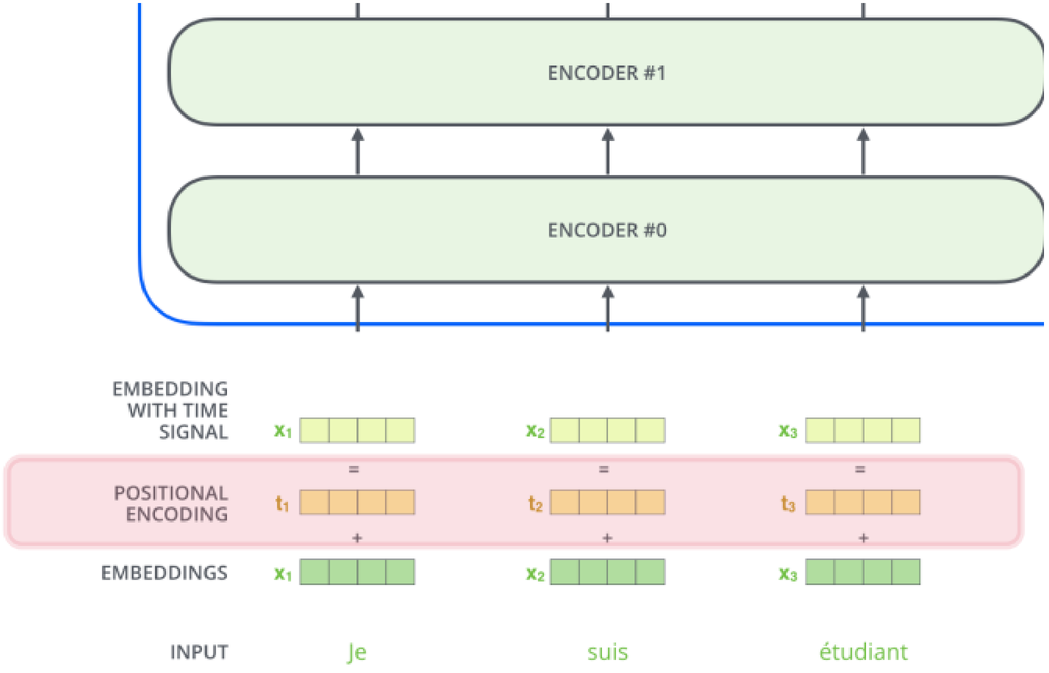
# Components of a Transformer

- To understand how to process graphs with Transformers, we must:
  - Understand the key components of the Transformer. Seen already:
    - 1) tokenizing,
    - 2) self-attention
  - Decide how to make suitable graph versions of each



# Components of a Transformer: Positional Encoding

- Transformer doesn't know order of inputs
- Extra positional features needed so it knows that
  - Je = word 1
  - suis = word 2
  - etc.
- For NLP, positional encoding vectors are learnable parameters

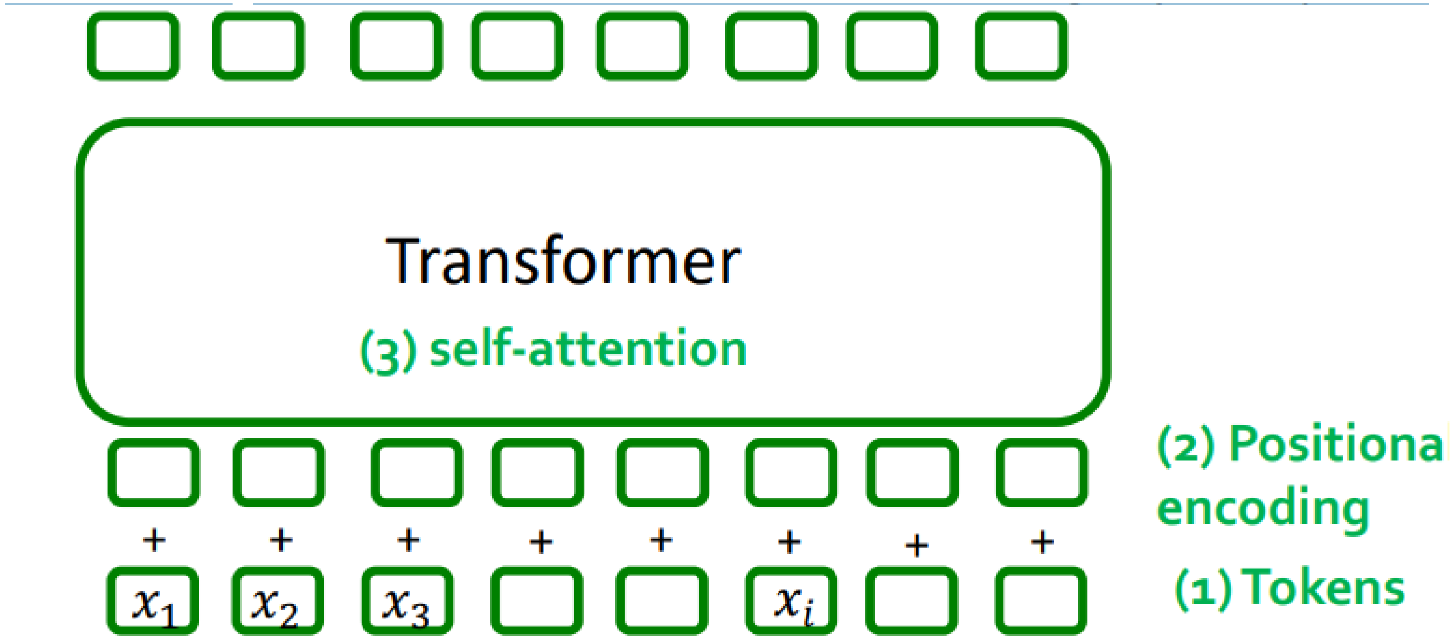


# Components of a Transformer



- Key components of Transformer:
  - (1) tokenizing
  - (2) positional encoding
  - (3) self-attention
- Key question: What should these be for a graph input?

How to chose these for graph data?



# Processing Graphs with Transformers

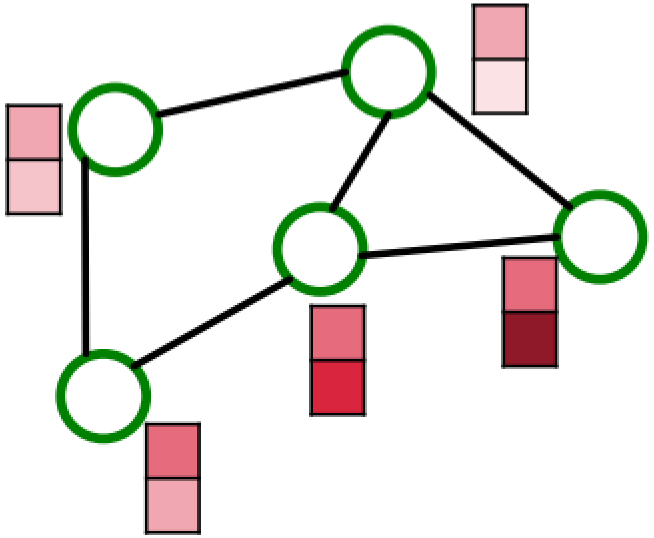


- A graph Transformer must take the following inputs:
  - (1) Node features?
  - (2) Adjacency information?
  - (3) Edge features (if any)

- Key components of Transformer:
  - (a) tokenizing
  - (b) positional encoding
  - (c) self-attention

**SOLUTIONS:**

- There are many ways to do this:
- Different approaches correspond to different "matchings" between graph inputs (1), (2), (3) transformer components (a), (b), (c)



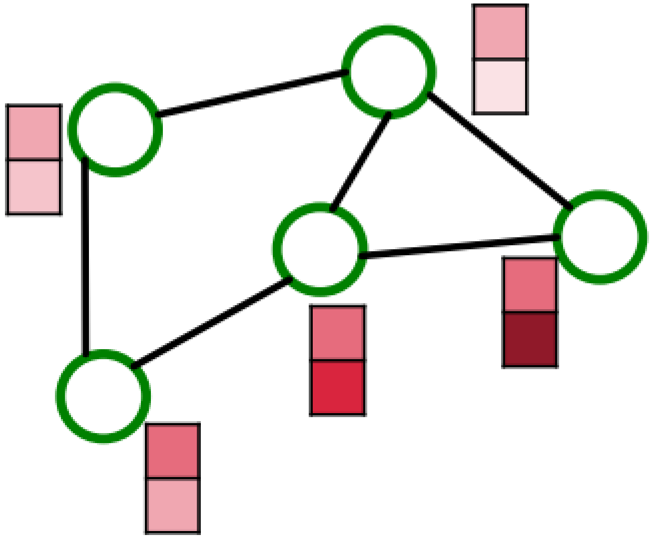
# Processing Graphs with Transformers

- A graph Transformer must take the following inputs:
  - (1) Node features?
  - (2) Adjacency information?
  - (3) Edge features (if any)

- Key components of Transformer:
  - (a) tokenizing
  - (b) positional encoding
  - (c) self-attention

**SOLUTIONS:**

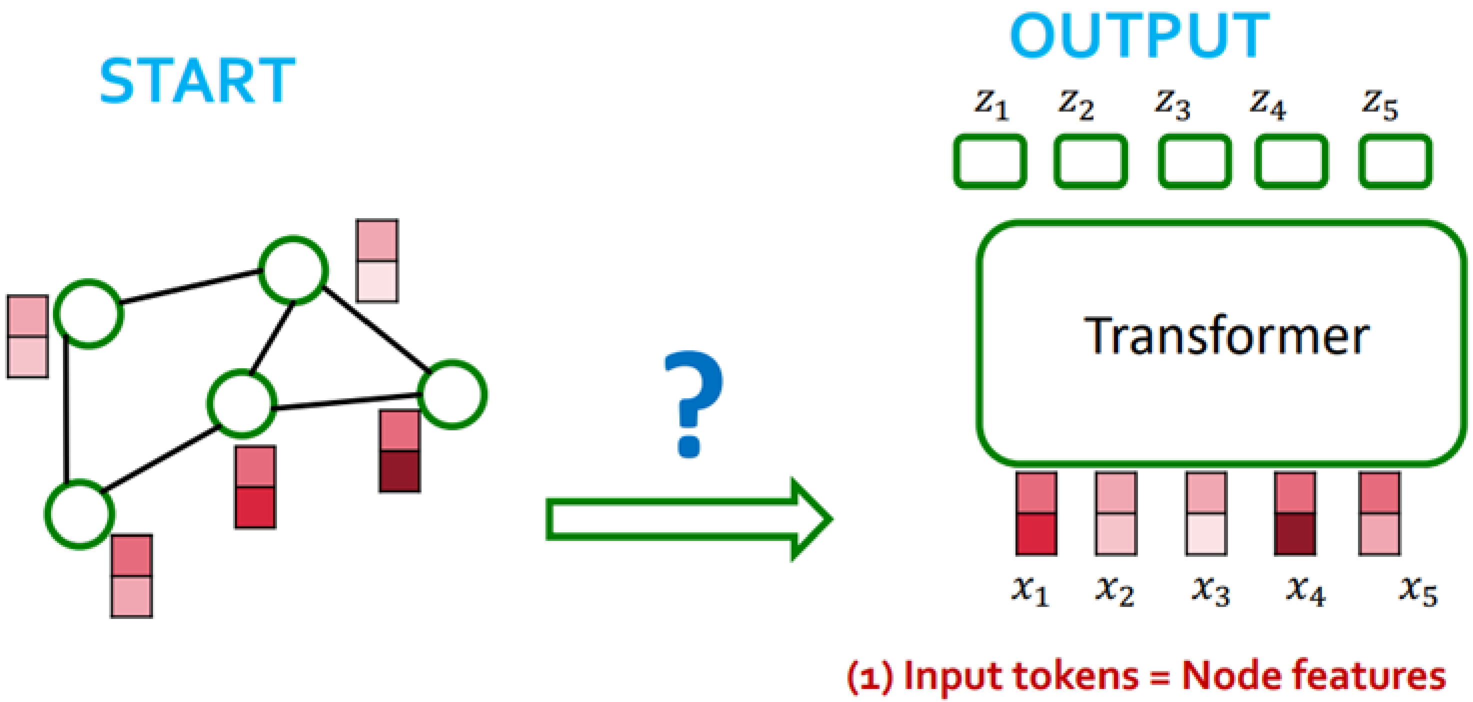
- There are many ways to do this:
- Different approaches correspond to different "matchings" between graph inputs (1), (2), (3) transformer components (a), (b), (c)



# Nodes as Tokens



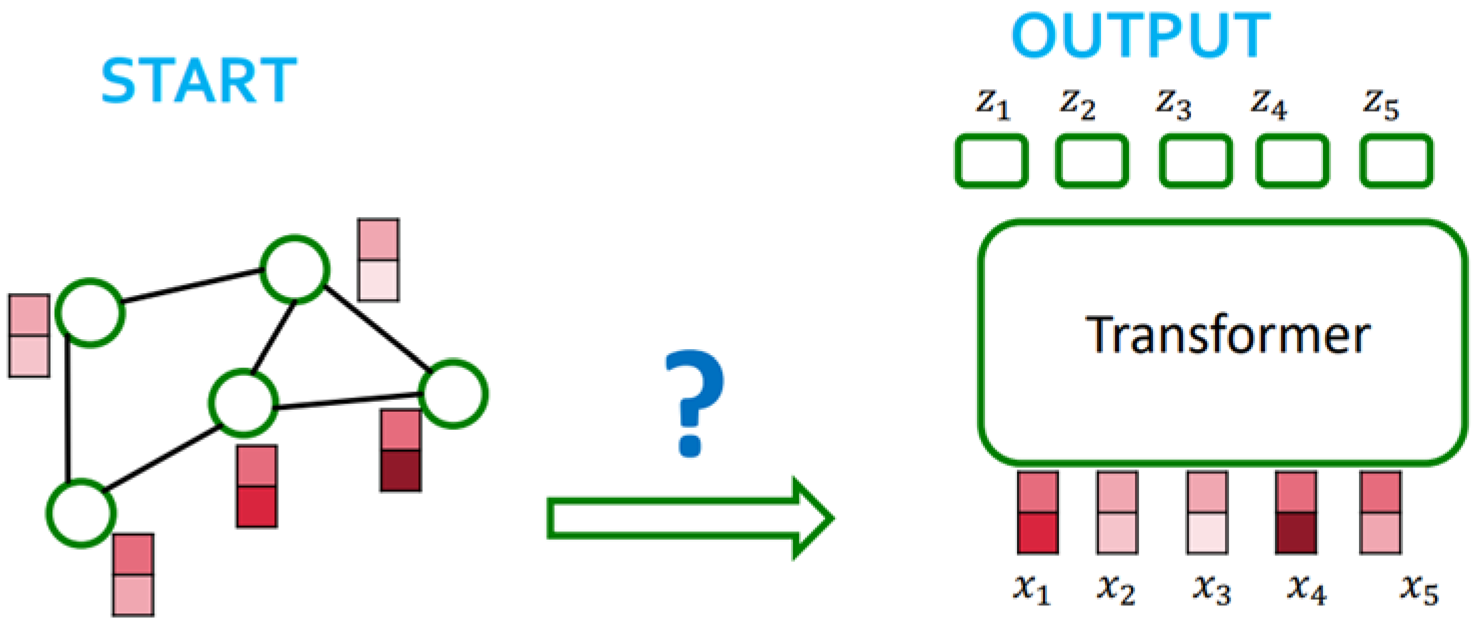
- **Q1**: what should our tokens be?
- **Sensible Idea:** node features = input tokens
- This matches the setting for the “attention is message passing on the fully connected graph” observation



# Nodes as Tokens



- Q1: what should our tokens be?
- Sensible Idea: node features = input tokens
- This matches the setting for the “attention is message passing on the fully connected graph” observation
- **Problem**? We completely lose adjacency info!
- **How to also inject adjacency information?**

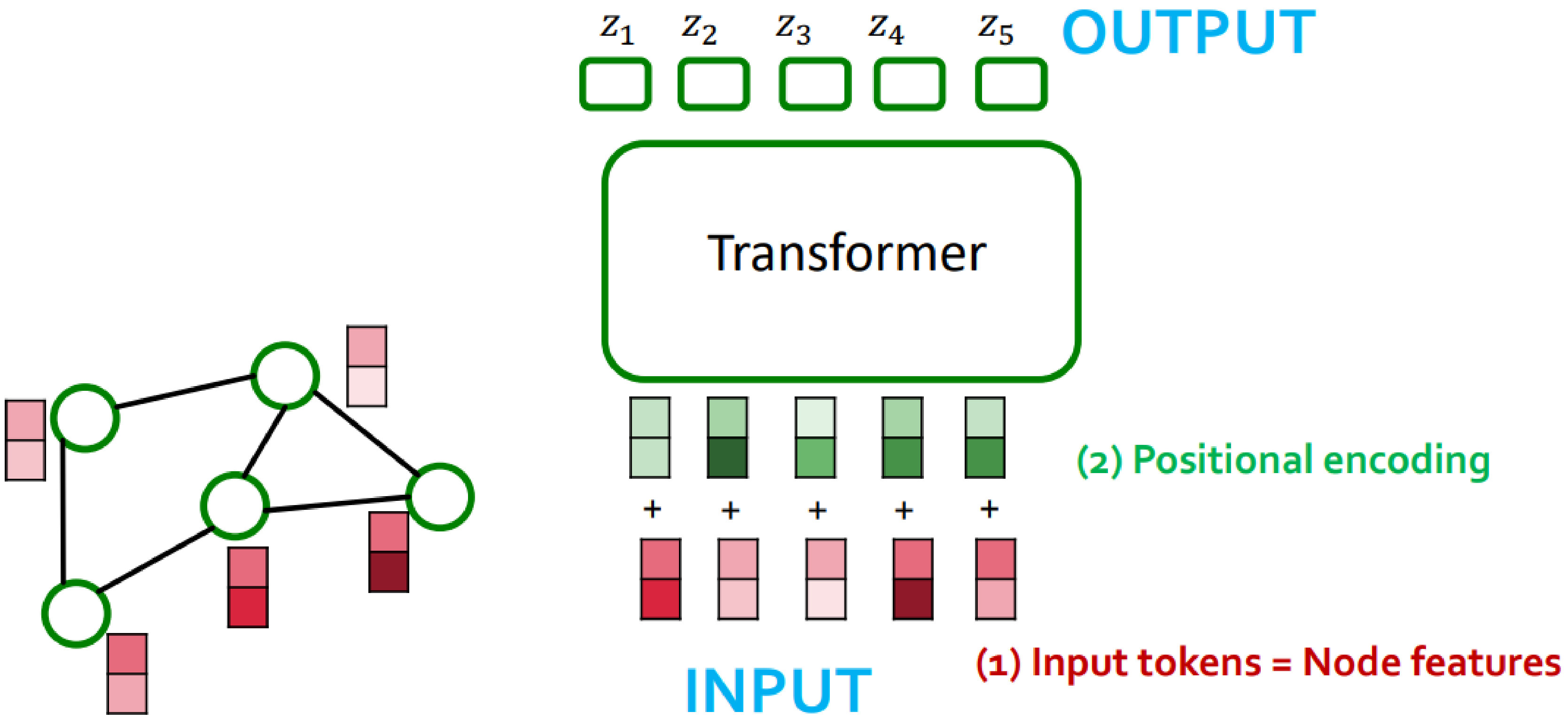



(1) Input tokens = Node features

# Nodes as Tokens: How to add back Adjacnecy information



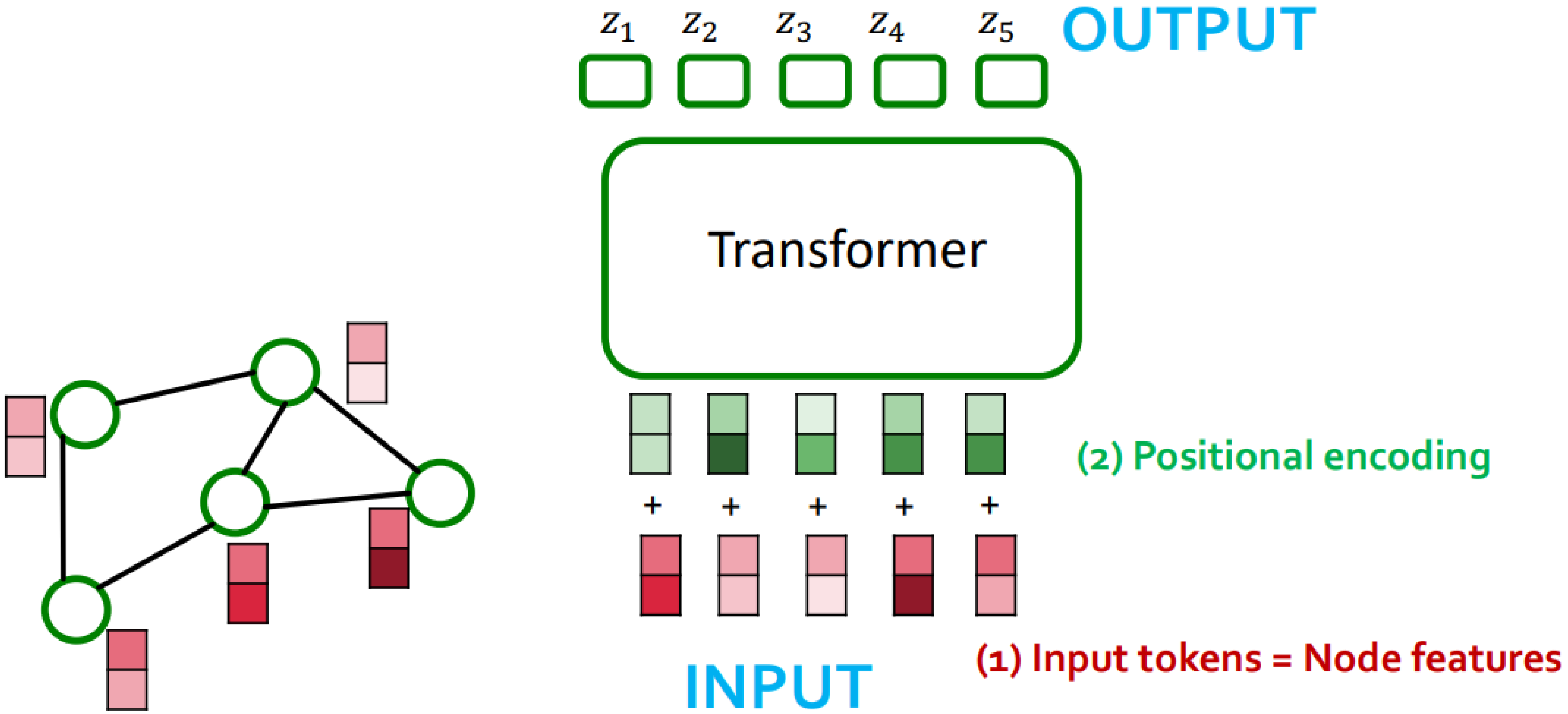
- **Problem**? We completely lose adjacency info!
- **How to also inject adjacency information?**
- **Idea:** Encode adjacency info in the positional encoding for each node
- Positional encoding describes where a node is in the graph



# Nodes as Tokens: How to add back Adjacnecy information

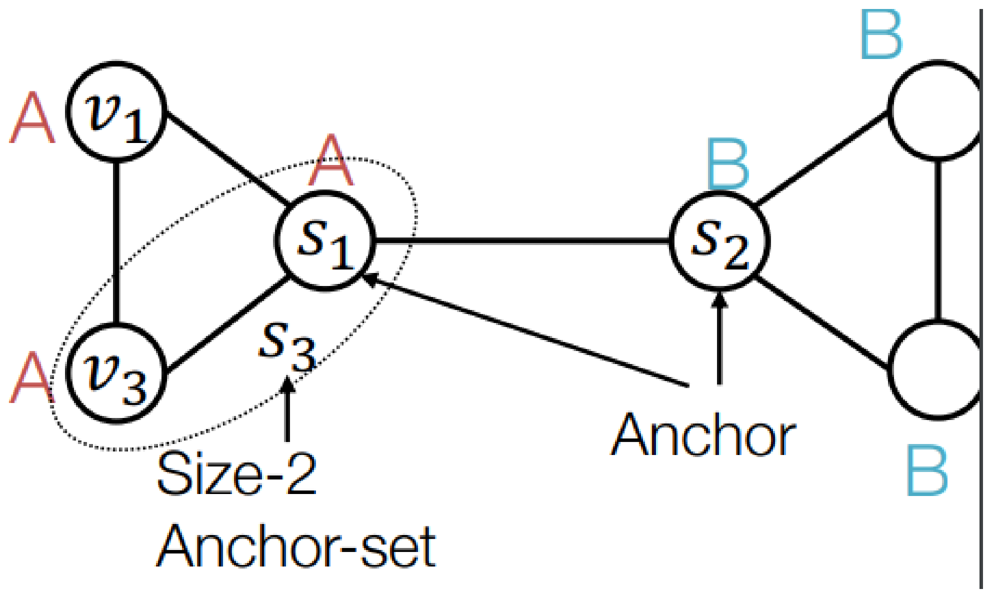


- **Q2: How to design a good positional encoding?**
  - Option 1: relative distance
  - Option 2: Laplacian Eigenvector PE



# Nodes as Tokens: Relative distance PE

- Similar methods based on random walks
- This is a good idea. It works well in many cases
- Especially strong for tasks that require counting cycles




Positional encoding for node $v_1$ =

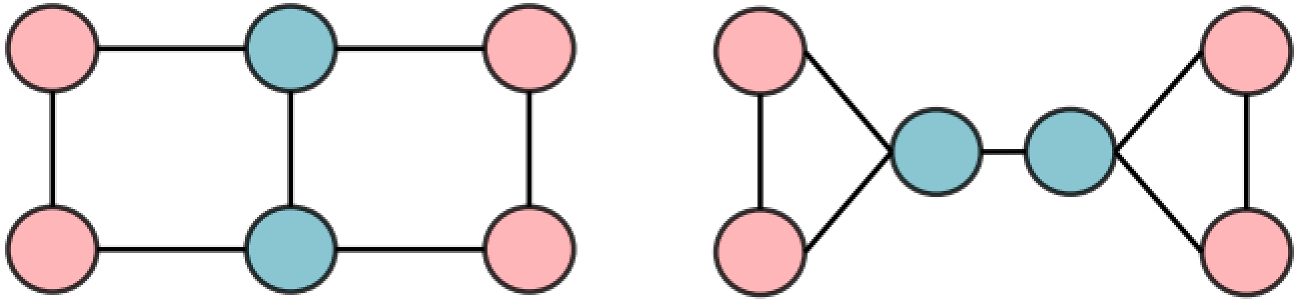
**Relative Distances**

| | $s_1$ | $s_2$ | $s_3$ |
|---|---|---|---|
| $v_1$ | 1 | 2 | 1 |
| $v_3$ | 1 | 2 | 0 |

Anchor $s_1$, $s_2$ cannot differentiate node $v_1$, $v_3$, but anchor-set $s_3$ can

# Nodes as Tokens: Relative distance PE

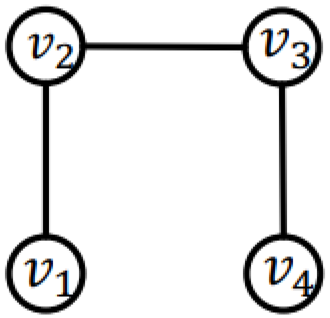
- Relative distances useful for position-aware task
- SPD can be used to improve WL-Test:
  - These two graphs cannot be distinguished by 1-WL-test.
  - But the SPD sets, i.e., the SPD from each node to others, are different:
  - The two types of nodes in the left graph have SPD sets {0, 1, 1, 2, 2, 3} , {0, 1, 1, 1, 2, 2} while the nodes in the right graph have SPD sets {0, 1, 1, 2, 3, 3} , {0, 1, 1, 1, 2, 2}.



# Nodes as Tokens: Laplacian Eigenvectors PE



- Draw on knowledge of Graph Theory (many useful and powerful tools)
- Key object: Laplacian Matrix L = Degrees - Adjacency
  - Each graph has its own Laplacian matrix
  - Laplacian encodes the graph structure
  - Several Laplacian variants that add degree information differently




$$L = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Degree of each node}} - \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Adjacency}}$$

# Nodes as Tokens: Laplacian Eigenvectors PE

- Laplacian matrix captures graph structure
- Its eigenvectors inherit this structure
- This is important because eigenvectors are vectors and so can be fed into a Transformer
- Eigenvectors with small eigenvalue = local structure, large eigenvalue = global symmetries

Refresher

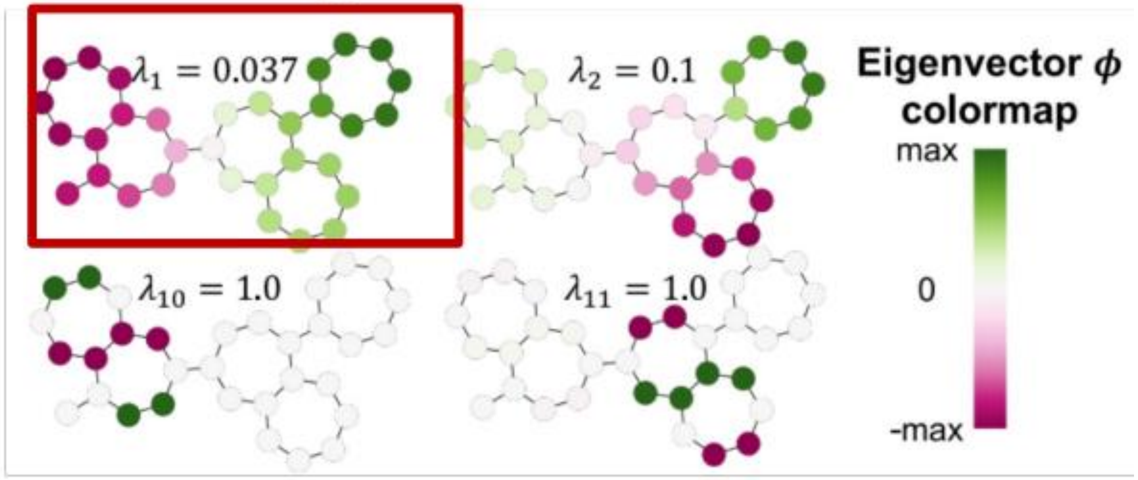
$\boxed{\text{Eigenvector: } v \text{ such that } Lv = \lambda v}$

$L: n \times n$ matrix

$v: n$ dimensional vector

$\lambda$: Scalar eigenvalue

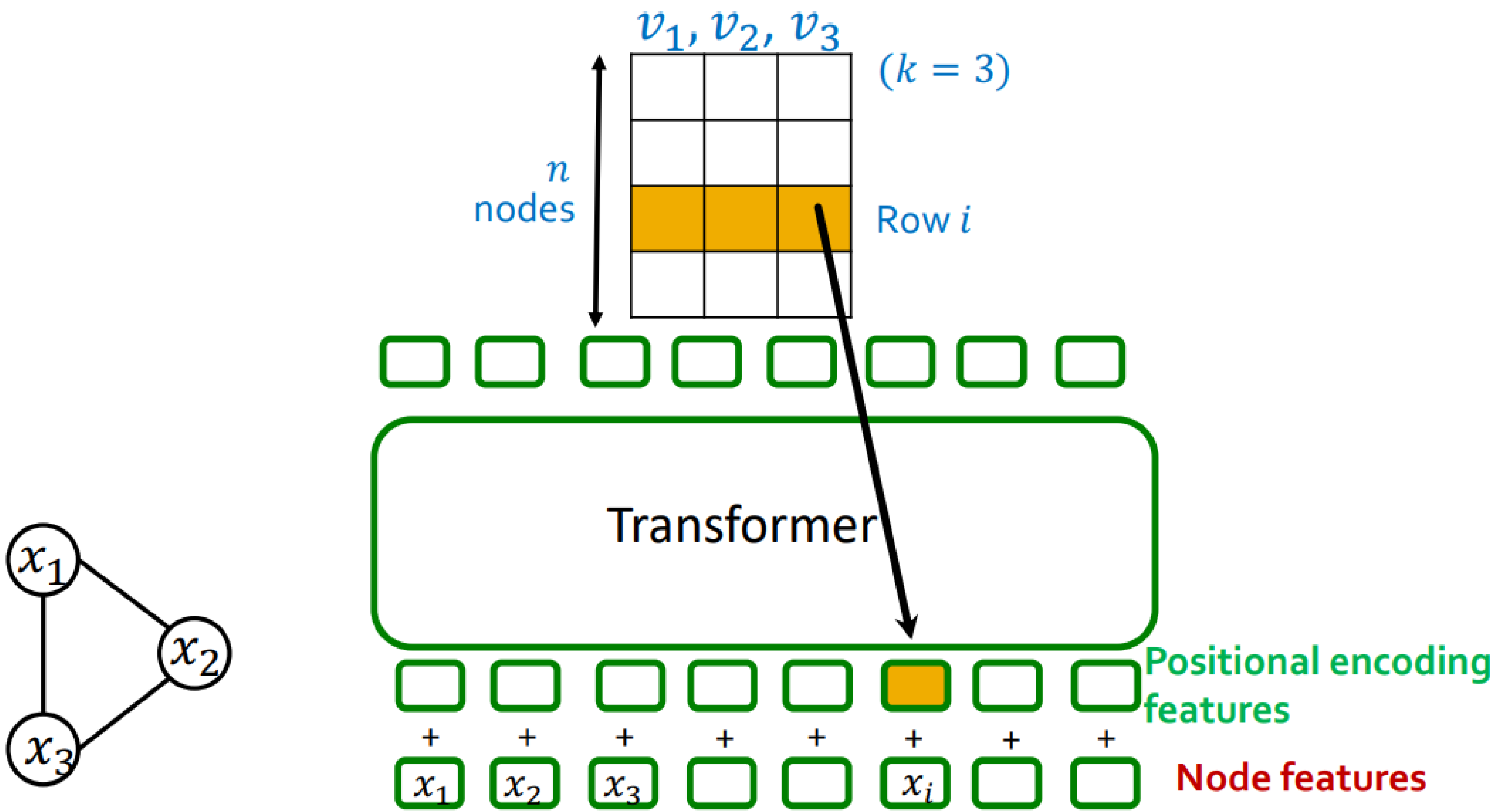
Visualize one eigenvector




(Figure from Kreuzer* and Beaini* et al. 2021)

# Laplacian Eigenvectors PEs

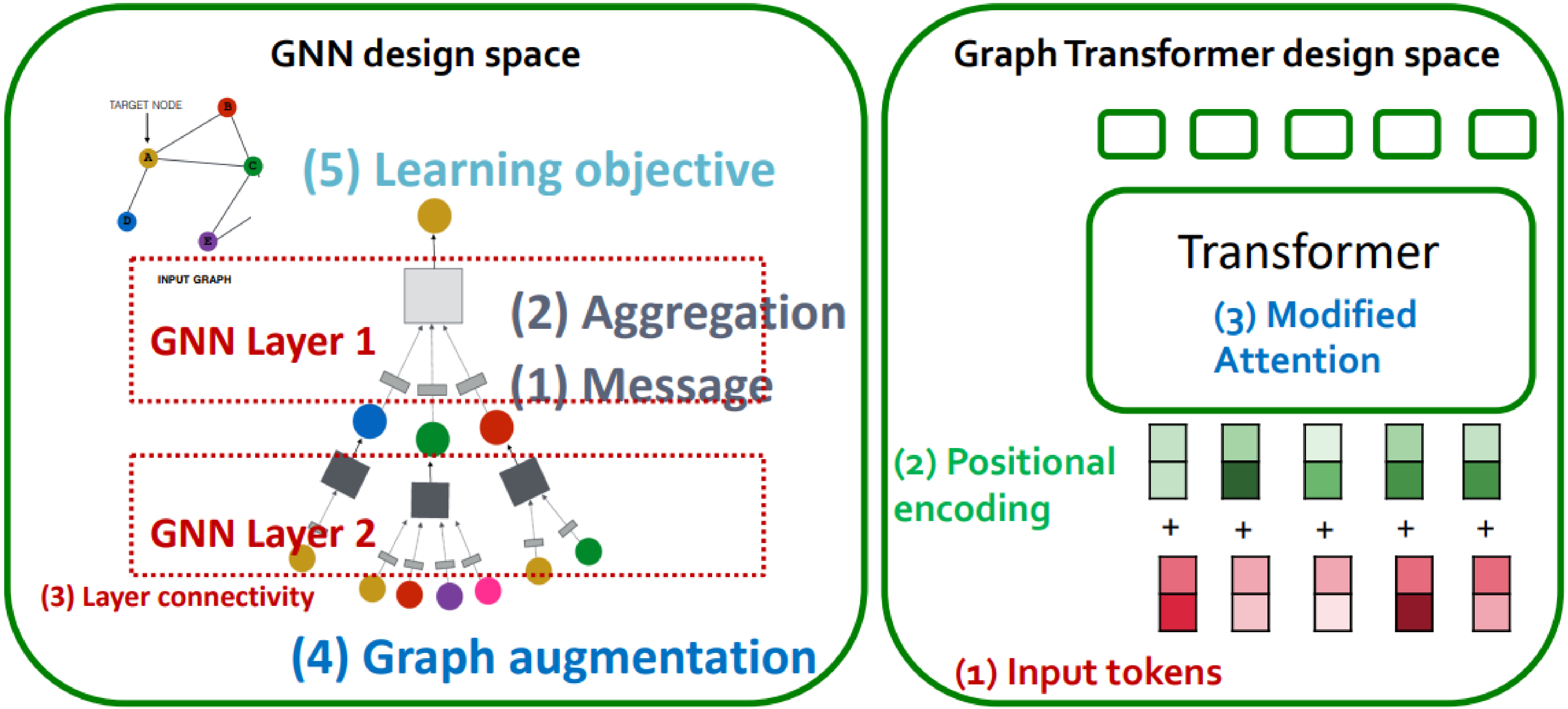
- Positional encoding steps:
  - 1. compute *k* eigenvectors
  - 2. Stack into matrix:
  - $i$-th row is positional encoding for node $i$



# Summary: Comparing Transformer and GNNs



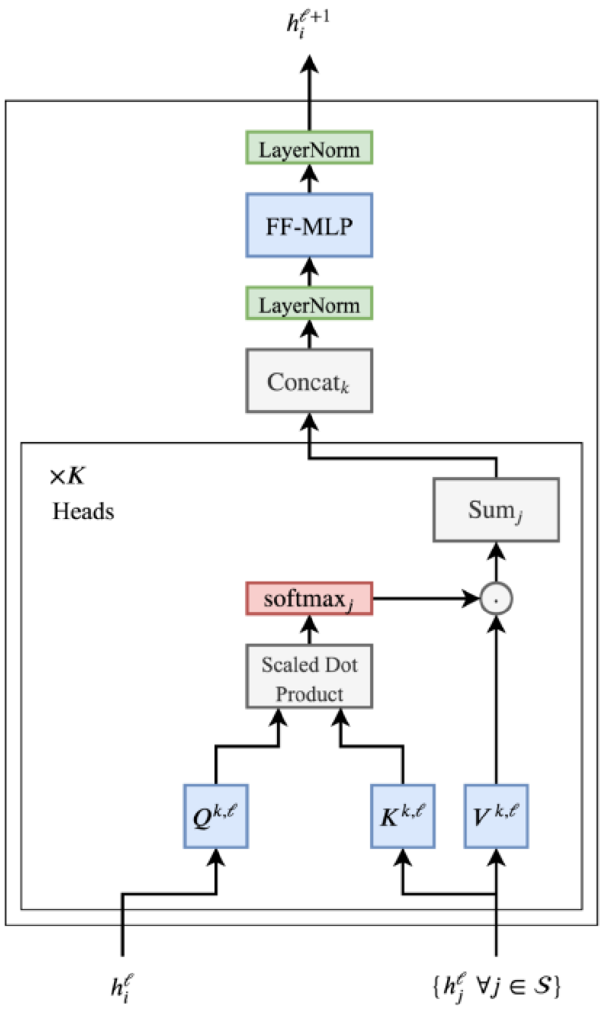
- Transformer are GNNs




https://web.stanford.edu/class/cs224w/slides/14-graph-transformer.pdf

# Why Transformers are GNNs?

- **Breaking down the Transformer**: Update each node's features through Multi-head Attention mechanism as a weighted sum of features of other words in the sentence.
  - Scaling dot product attention
  - Normalization layers
  - Residual links



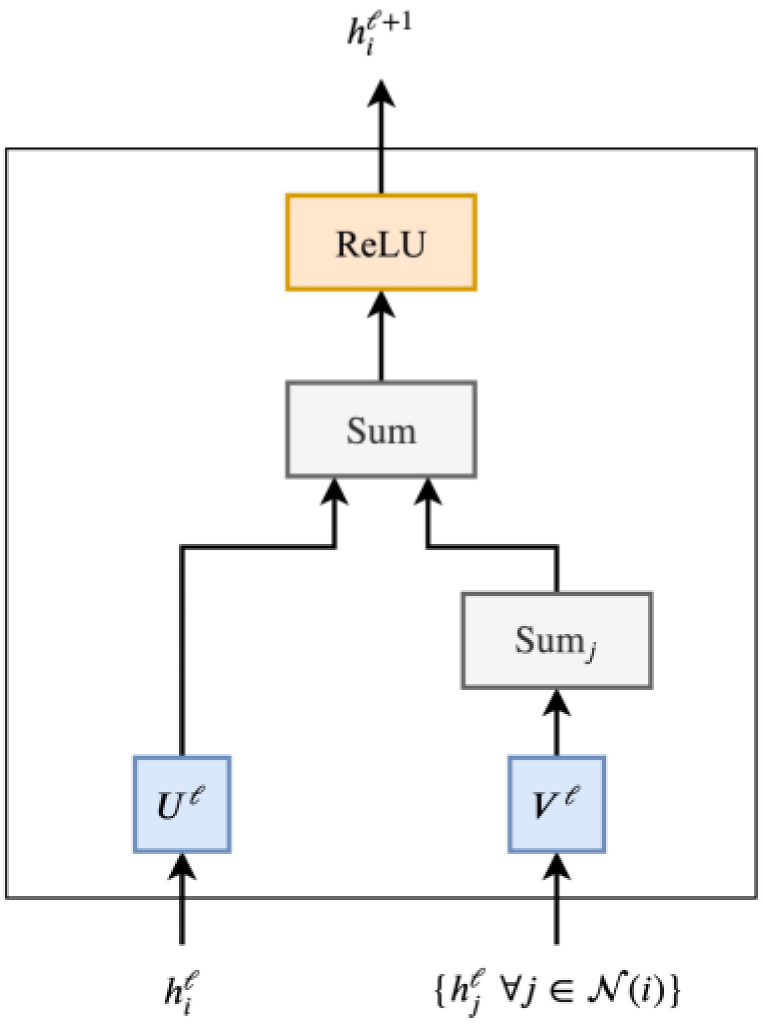

Transformer

# Why Transformers are GNNs?

- **Breaking down the GNNs**: GNNs update the hidden features $h$ of node $i$ at layer $l$ via a non-linear transformation of the node's own features added to the aggregation of features from each neighbouring node $j \in N(i)$:

$$h_i^{\ell+1} = \sigma\Big( U^{\ell} h_i^{\ell} + \sum_{j \in \mathcal{N}(i)} \left( V^{\ell} h_j^{\ell} \right) \Big),$$

- where U, V are learnable weight matrices of the GNN layer and $\sigma$ is a non-linearity.




GNNs

# Why Transformers are GNNs?

- **Breaking down the Transformer and GNNs**:

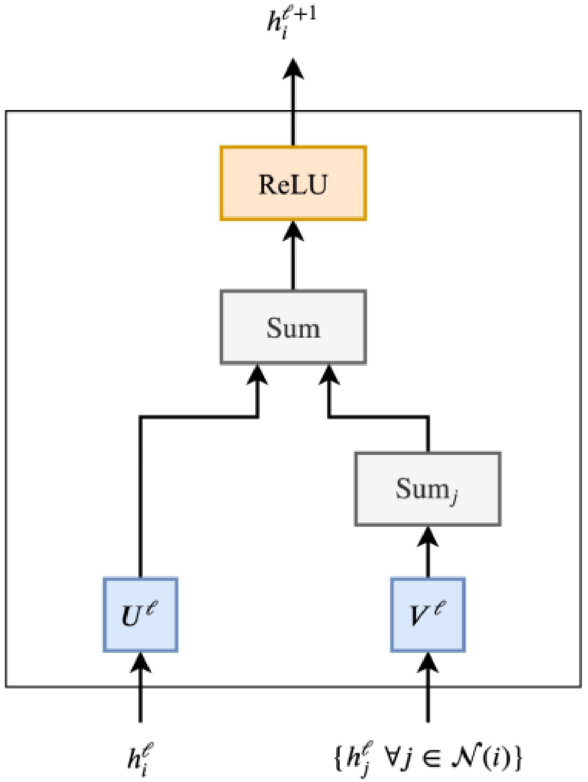
- **GNNs**:

$$h_i^{\ell+1} = \sigma\Big(U^\ell h_i^\ell + \sum_{j \in \mathcal{N}(i)} (V^\ell h_j^\ell)\Big),$$

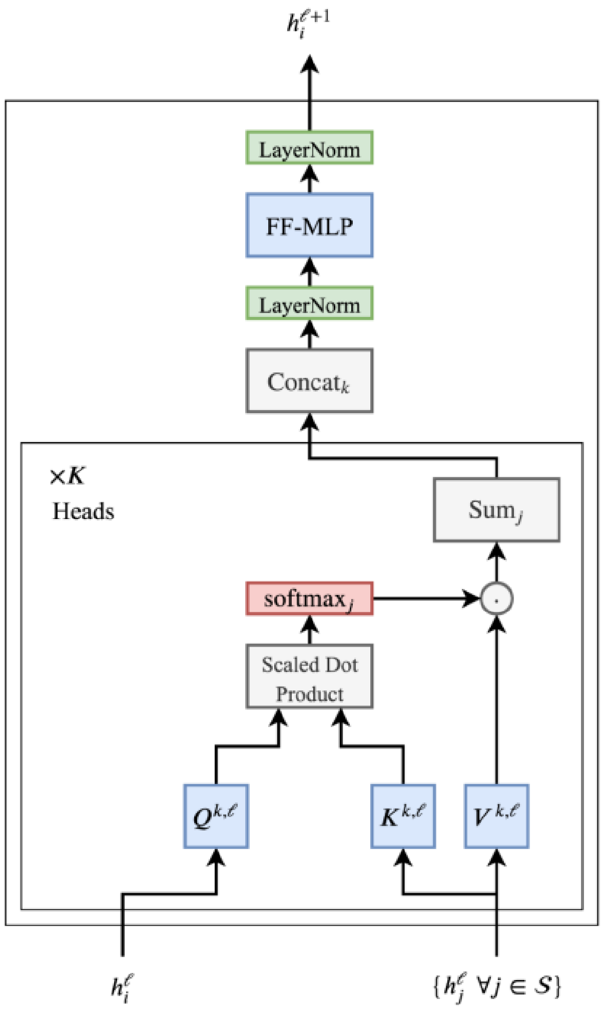
- **Transformers**:

$$i.\ e.\ ,\ h_i^{\ell+1} = \sum_{j \in \mathcal{S}} w_{ij}(V^\ell h_j^\ell),$$

$$\text{where } w_{ij} = \text{softmax}_j(Q^\ell h_i^\ell \cdot K^\ell h_j^\ell),$$




GNNs



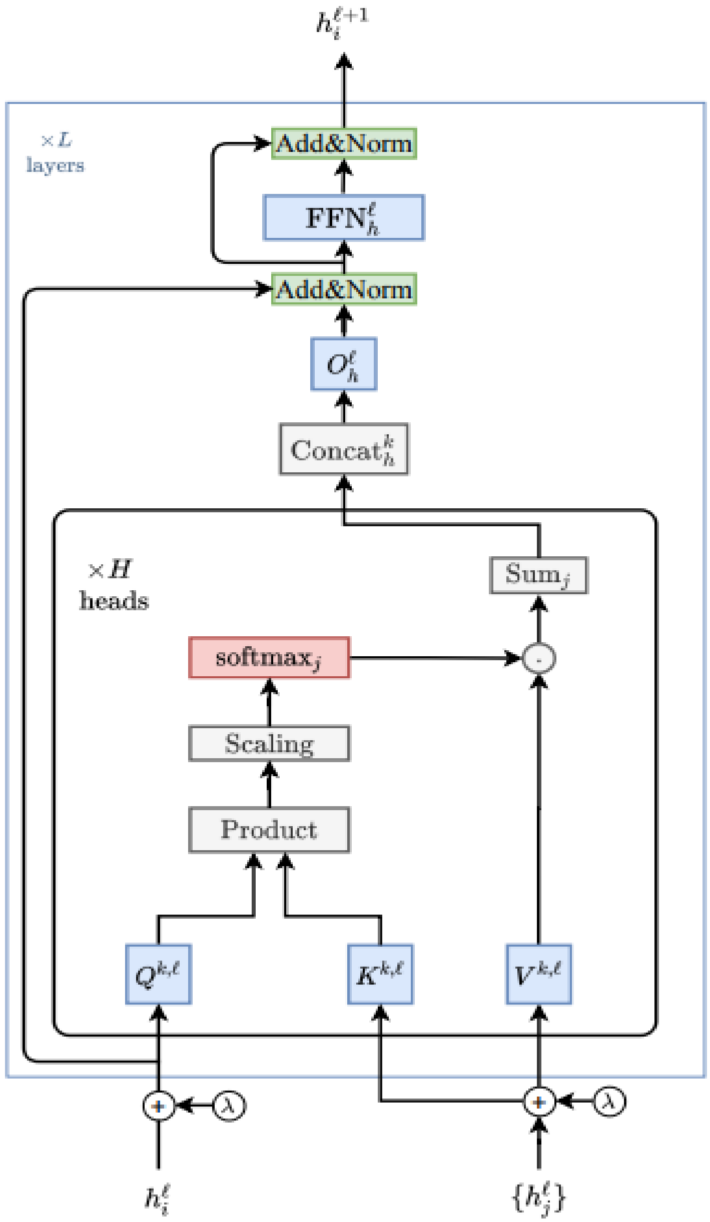

Transformer

**GT (Graph Transformers *)**

- Using Laplacian Eigvectors (λ) used as positional encoding (LapPE).
- Graph Transformer Layer:

$$\hat{h}_i^{\ell+1} = O_h^{\ell} \Big\Vert_{k=1}^{H} \Big( \sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \Big),$$

$$\text{where, } w_{ij}^{k,\ell} = \text{softmax}_j \left( \frac{Q^{k,\ell} h_i^{\ell} \cdot K^{k,\ell} h_j^{\ell}}{\sqrt{d_k}} \right)$$




* A Generalization of Transformer Networks to Graphs, AAAI 2021

# Representative Transformer models : Graphormer

**Graphormer (*)**

- Centrality Encoding:

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)},$$

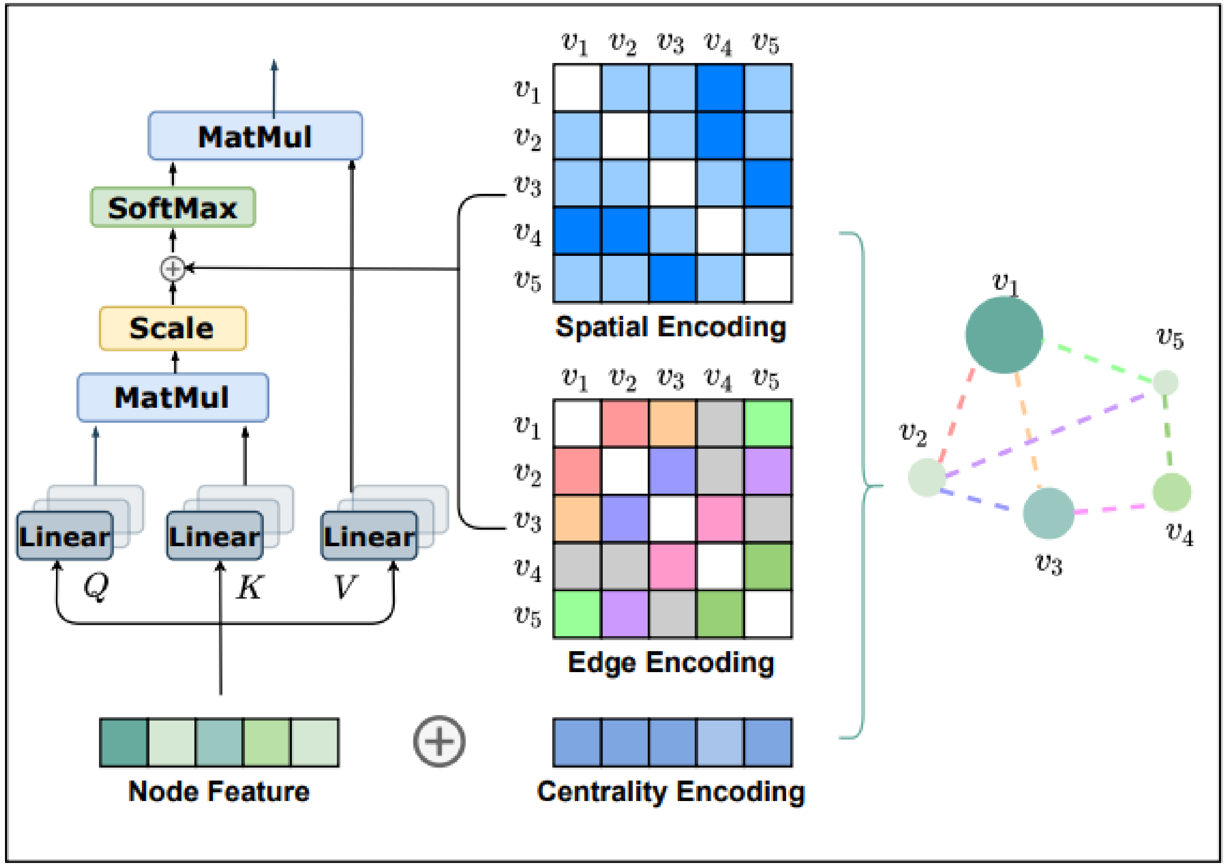
(learnable indegree $z^-$, and outdegree $z^+$)

- Self-attention bias:

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i,v_j)} + c_{ij}$$

$$\text{where } c_{ij} = \frac{1}{N}\sum_{n=1}^{N} x_{e_n}(w_n^E)^T$$

presents the path between two nodes $i$ and $j$ via edge feature path: $\mathrm{SP}_{ij} = (e_1, e_2, ..., e_N)$



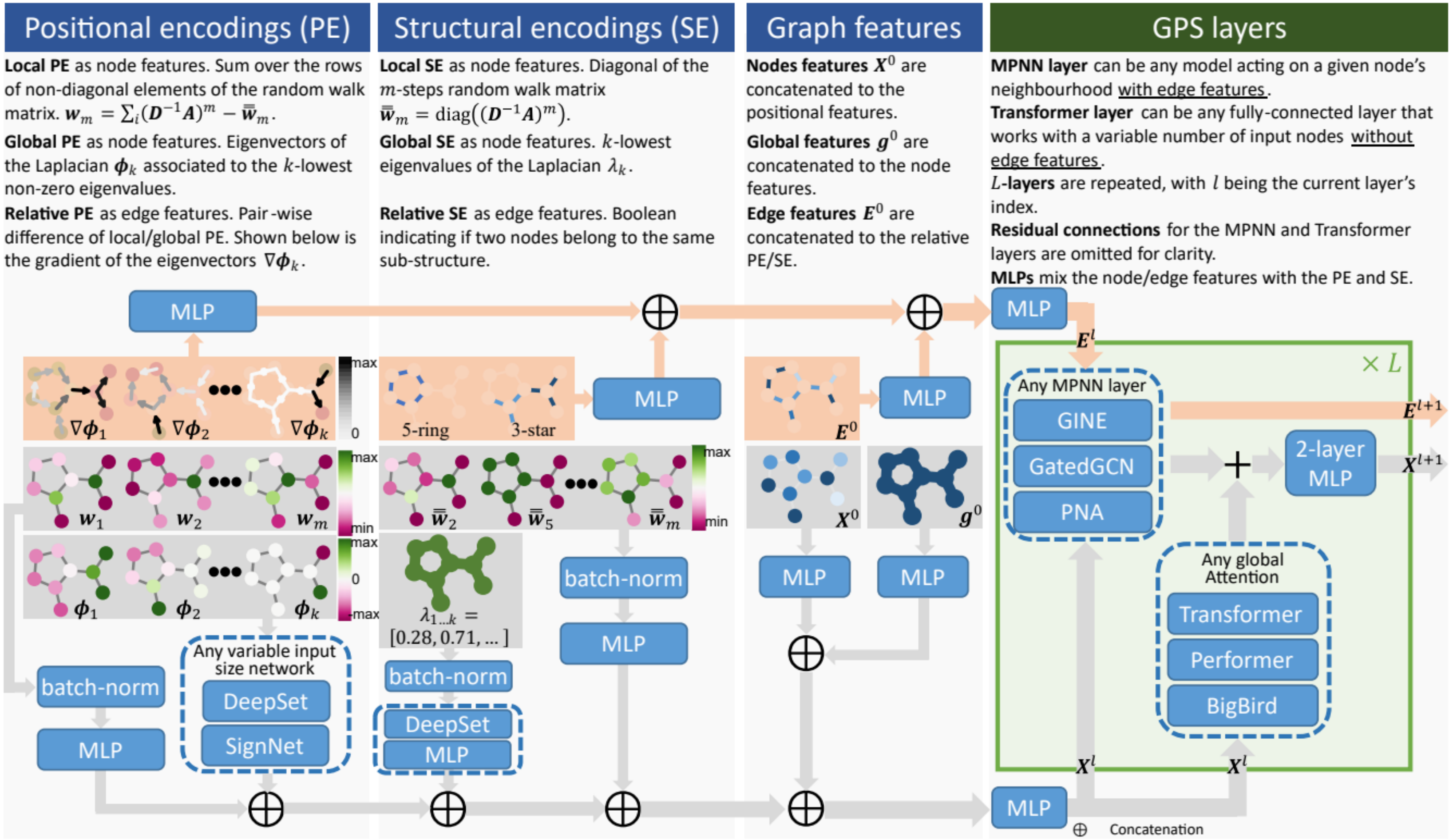

$b_{\phi(v_i,v_j)}$: the distance of the shortest path (SPD) between two nodes $i$ and $j$

* A Do Transformers Really Perform Bad for Graph Representation? (NeurIPS 2021)

# Representative Transformer models : GPS

- **GPS uses:**
  - Randomwalk PE
  - GPS layers:
    - An MPNN+
    - Transformer hybrid

$$
\begin{aligned}
\mathbf{X}^{\ell+1}, \mathbf{E}^{\ell+1} &= \mathtt{GPS}^{\ell}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}\right) \\
\text{computed as} \quad \mathbf{X}_M^{\ell+1}, \mathbf{E}^{\ell+1} &= \mathtt{MPNN}_e^{\ell}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}\right), \\
\mathbf{X}_T^{\ell+1} &= \mathtt{GlobalAttn}^{\ell}\left(\mathbf{X}^{\ell}\right), \\
\mathbf{X}^{\ell+1} &= \mathtt{MLP}^{\ell}\left(\mathbf{X}_M^{\ell+1} + \mathbf{X}_T^{\ell+1}\right),
\end{aligned}
$$

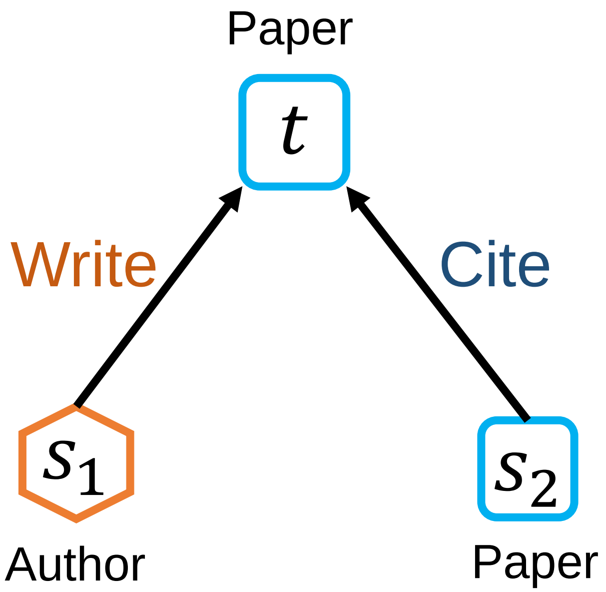


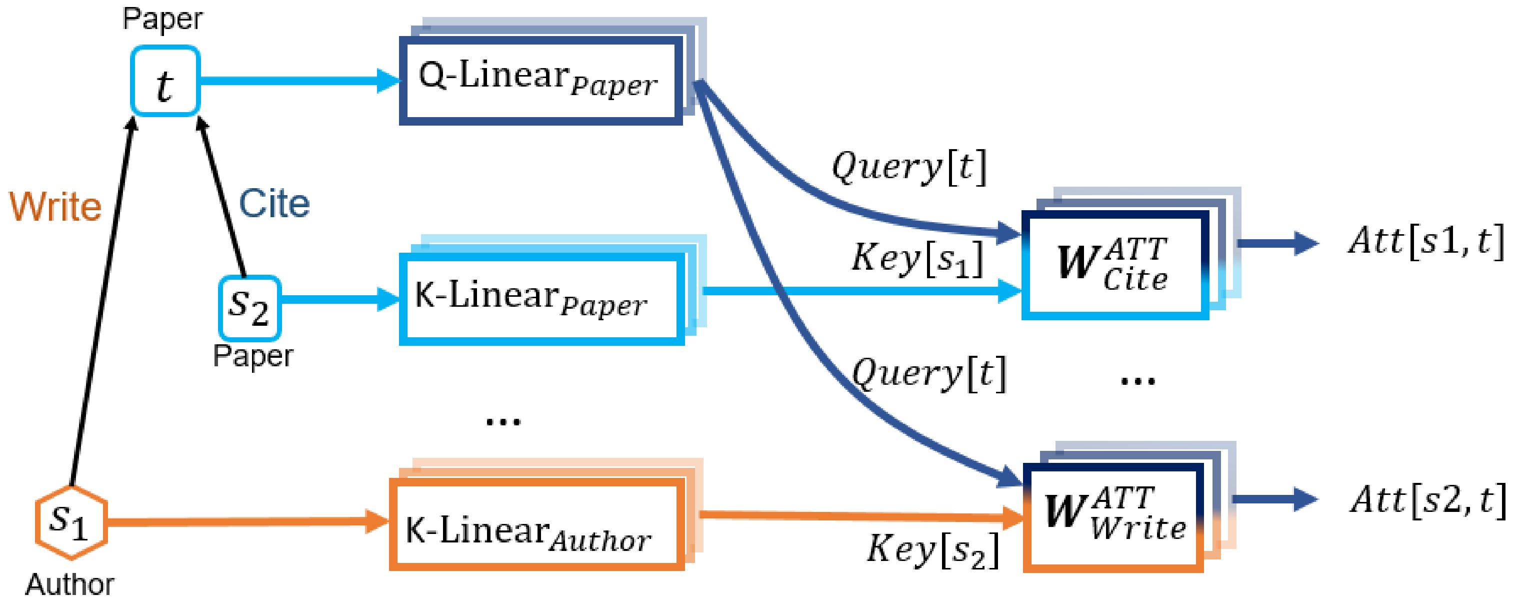

GPS: a General, Powerful, Scalable graph Transformer

Recipe for a General, Powerful, Scalable Graph Transformer (NeurIPS 2022)

- Heterogeneous Mutual Attention in heterogeneous Graphs

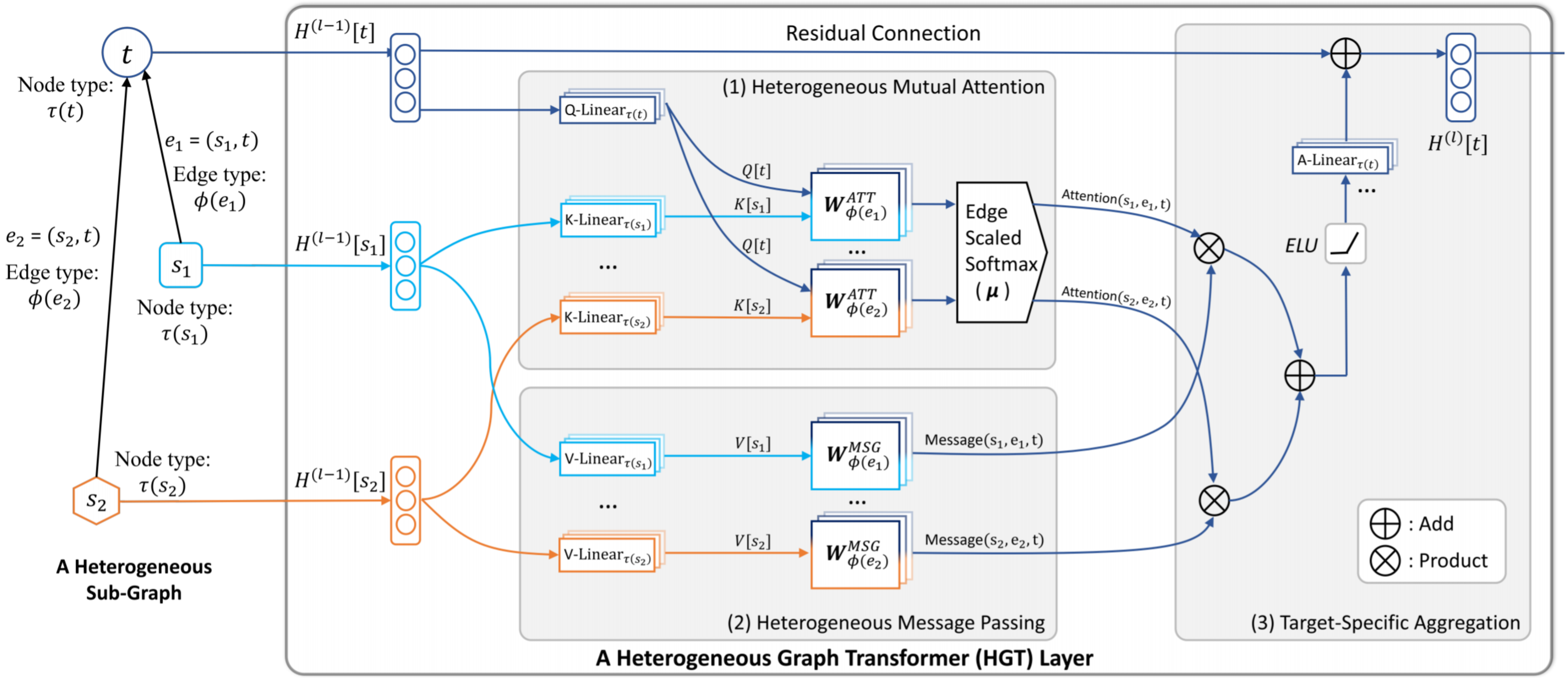
$W_{\langle Author, Write, Paper\rangle}$
$= W_{Author}\ W_{Write}\ W_{Paper}$




$W_{\langle Paper, Cite, Paper\rangle}$
$= W_{Paper}\ W_{Cite}\ W_{Paper}$

# Representative: Heterogeneous Graph Transformer




Heterogeneous Graph Transformer; WWW ‘2020

Q A

네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA